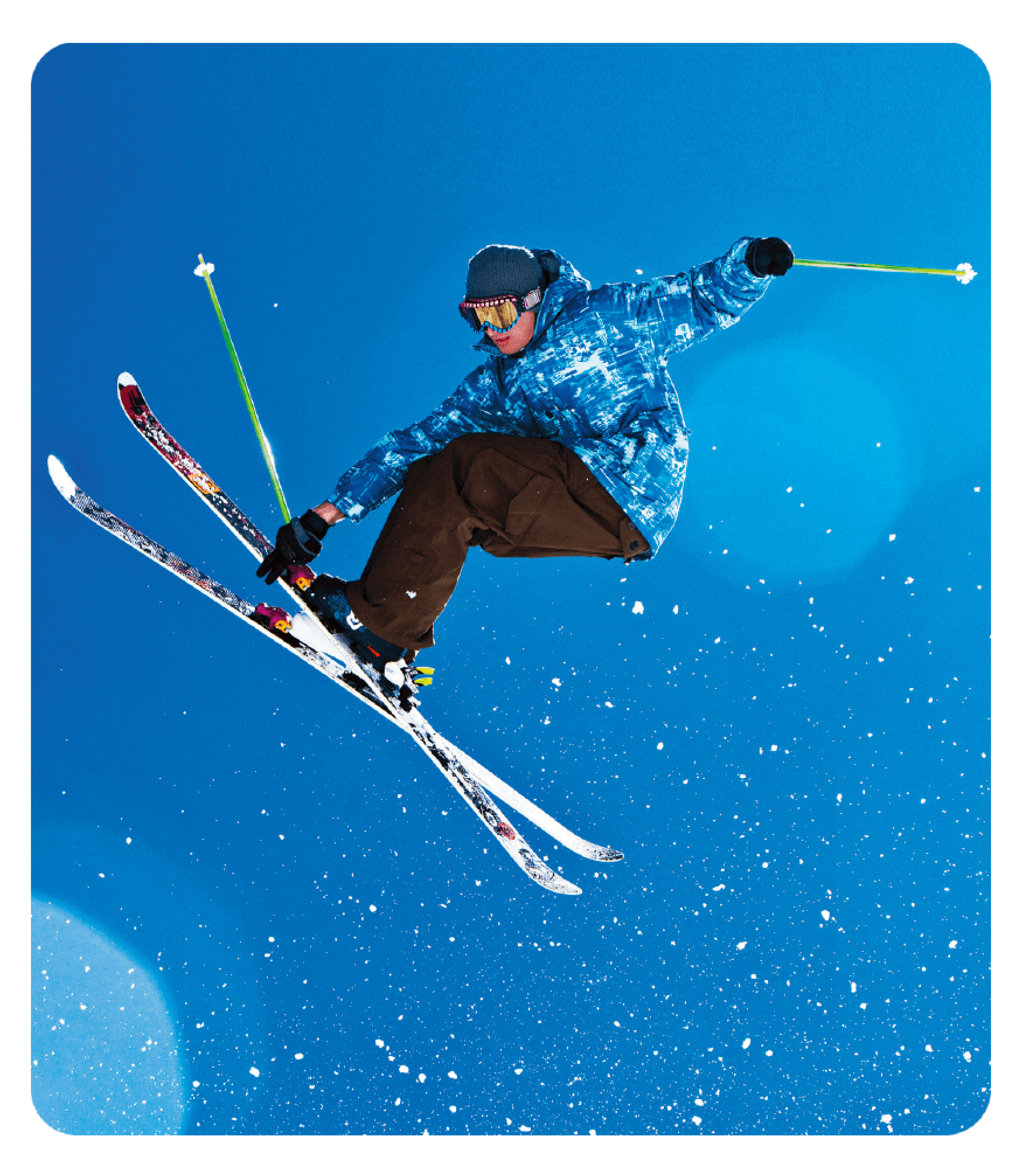
慧与国际软件人才及产业基地慧据教育系列丛书

Web 前端开发技术
——HTML5+Ajax+jQuery

张　朋　编著



清华大学出版社

慧与国际软件人才及产业基地慧据教育系列丛书

Web 前端开发技术
—— HTML5+Ajax+jQuery

张 朋 编著

清华大学出版社
北 京

内 容 简 介

本书介绍了 Web 前端开发需要的各种技术，如 HTML5、CSS3、JavaScript、jQuery、jQuery Mobile，以及多媒体新特性、HTML5 浏览器兼容性测试、DOM 模型、Ajax 原理、JSON 格式等。本书是作者多年教学、项目经验的总结，汇集了学生在学习 Web 前端技术时遇到的各种问题及解决方案，在介绍范例代码的时候，使用了较新的开发工具及在线编辑工具，使读者能够高效地完成书中的案例。

本书适合 Web 前端技术初学者学习，也适合软件培训机构采纳作为培训教材，可作为应用类本科及高职高专学生 Web 课程的教材。

图书在版编目(CIP)数据

Web 前端开发技术——HTML5+Ajax+jQuery / 张朋 编著. —北京：清华大学出版社，2017
(慧与国际软件人才及产业基地慧据教育系列丛书)
ISBN 978-7-302-44807-5

Ⅰ. ①W… Ⅱ. ①张… Ⅲ. ①超文本标记语言—程序设计②计算机网络—程序设计③JAVA 语言—程序设计 Ⅳ. ①TP312②TP393.09

中国版本图书馆 CIP 数据核字(2016)第 192649 号

责任编辑： 王 军 李维杰
装帧设计： 牛静敏
责任校对： 成风进
责任印制：

出版发行： 清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦 A 座　　邮 编：100084
社 总 机：010-62770175　　邮 购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者： 北京九州迅驰传媒文化有限公司
经 销： 全国新华书店
开 本： 185mm×260mm　　**印 张：** 19.5　　**字 数：** 487 千字
版 次： 2017 年 7 月第 1 版　　**印 次：** 2017 年 7 月第 1 次印刷
定 价： 56.00 元

产品编号：

前　　言

惠普在 2015 年 11 月进行了战略拆分，正式拆分成两家独立企业，其中新成立“Hewlett Packard Enterprise (简称为 HPE)”，专注于企业级业务，核心业务就是领先的软件资产，由惠普领先的企业技术基础架构业务、软件业务和服务业务组成；另一家为 HP Inc，由惠普领先的个人系统业务和打印业务组成。

2016 年 2 月 1 日起，HPE 正式发布了新的品牌名称及品牌 LOGO。HPE 中文品牌名称“慧与”应运而生。“慧”象征智慧，和“惠”同样的发音联结着我们与过去的传承与发展。“与”寓意合作，表达我们与员工、合作伙伴、客户间携手共赢的关系。

慧与国际软件人才及产业基地开展实训业务三年来，在全国几十所高校设立了共建专业，并与全国几千家企业建立了人才合作关系，培养了近万名软件人才。其中软件开发方向开设了 JavaEE、大数据、.NET、Android 等专业，处于行业领先水平。

近年来，随着互联网的兴起以及传统企业与互联网的结合，前端 Web 系统的复杂性不断增加，出现了许多专职的前端职位，比如高级前端开发工程师、前端架构师。Web 网站是一个从前到后的系统工程，从公司的角度看，实现这个系统需要的技术资源越来越多，包括系统设计、JavaScript、后端技术、UX、数据库。2010 年之后，后端技术更多迁移到云及大数据上，而其他技术大多涉及前端开发，以至于在 2010 年和 2014 年底出现了对 HTML5 工程师需求的爆发式增长。

Web 前端方面的工作由原来的重视 Web 设计，发展到 Web 设计、Web 开发、Web 交互、Web 性能并重，这无疑对 Web 前端行业的从业者提出了巨大挑战。现在网上有许多介绍前端开发知识的文章，但是前端从业者发现找到适合自己的知识却不容易。本书从全局角度，尽量覆盖一名前端开发工程师所需的技能，构建企业所需的技能体系。

全书分四大部分，内容涉及 HTML5、CSS3、MongoDB、JavaScript、jQuery 和 jQuery Mobile。

其中每部分包含多个章节。本书是一本涵盖 Web 前端开发基础知识的教程，不仅包括从事 Web 前端开发需要掌握的理论知识和技术详解，还包括企业项目的实战，分析前端技术的使用场景及其在项目中的作用。

本书内容

全书共分为 10 章，主要内容如下：

第 1 章：介绍 HTML5 的发展历史、HTML5 的新特性及其背后的设计理念，详细介绍 HTML5 的新元素，包括 HTML5 新的结构元素、分组元素和语义元素、多媒体元素和 HTML5 表单。

第 2 章：介绍 CSS 的新特性、BootStrap 的作用及其使用方法。

第 3 章：介绍 JavaScript 基础内容以及使用 JavaScript 创建交互式网页，从 JavaScript 的基本语句到对象和函数的高级用法，都列举了实例，还介绍了浏览器的事件处理方法。

第 4 章：JavaScript 图形编程是 HTML5 最吸引人的特性之一，本章由浅入深地介绍了画布的各种概念和使用方法，还穿插了多个实例，帮助读者理解画布的具体开发流程。

第 5 章：介绍 MongoDB 的基本概念、增删改查基本操作，使读者能够马上入门和体验 MongoDB 的好处。

第 6 章：介绍 jQuery 的核心内容，使用 jQuery 能用最少的代码实现最多的功能，还介绍如何遍历 HTML 文档、处理事件，并对浏览器 DOM 模型进行了补充介绍。

第 7 章：介绍 Ajax 异步通信原理，并根据这个原理分析 jQuery 是如何实现这些异步通信的。

第 8 章：介绍流行的 jQuery 插件，讨论 jQuery 的扩展能力以及如何寻找插件。

第 9 章：介绍 Ajax 的具体应用，使用 jQuery Mobile 开发移动端应用。

第 10 章：介绍 jQuery 最常见的动画效果，结合使用 jQuery 的 DOM 操作、CSS 操作、动画特效和制作出绚丽的页面。

目　　录

第1章　HTML5

2014 年 10 月 28 日，万维网联盟(W3C)宣布，历经 8 年的努力，HTML5 标准规范最终制定完成，并已公开发布。此前的几年时间里，已经有很多开发者陆续使用了 HTML5 的部分技术，Firefox、Google Chrome、Opera、Safari 4+、Internet Explorer 9+都已支持 HTML5。

HTML5 是近十年来 Web 开发标准最巨大的飞跃。和以前的版本不同，HTML5 并非仅用来表示 Web 内容，它的新使命是将 Web 带入一个成熟的应用平台。在 HTML5 平台上，视频、音频、图像、动画以及与计算机的交互都被标准化。HTML5 在这个时间定稿，不早不晚，正是硬件性能更强、手机 OS 迭代速度下降的时期。随着 HTML5 标准的定稿，一切纷争将告一段落。现在，属于 HTML5 的时代到来了。

本章内容：

- 掌握 HTML5 的文档结构
- 掌握 HTML5 新增的语义化元素
- 使用 HTML5 中的多媒体元素
- 掌握 HTML5 表单元素

1.1　HTML5 介绍

1.1.1　HTML5 发展史

互联网发展到 2005 年前后，开始出现了一个变化，就是宽带互联。随着宽带的普及和计算机性能的增强，人们不再满足于单纯地通过互联网看新闻、收发邮件，消耗更高带宽的娱乐产品开始出现，就是流视频和网页游戏。HTML 标准没有把握住产业的变化而及时演进，浏览器产品也未升级，这块新需求被浏览器插件满足了，那就是 Flash。这个部署在亿万浏览器里的商业插件俨然成为事实标准。但乔布斯力挺 HTML5，他坚持在 iOS 上不兼容 Flash。为了能够支持新的 Web 应用，同时克服现有的缺点，HTML 迫切需要添加新功能、制定新规范。

一小组人为了弥补 HTML 中最薄弱的环节，开始专门针对 Web 应用开发新功能，他们创立了 HTML5 规范。就是在那个时候开创了 Web 的第二个时代，Web 2.0 这个新词也被发明出来，这主要表现在旧的静态网站逐渐让位于需要更多特性的动态网站和社交网站。2006 年，W3C 又重新介入 HTML，并于 2008 年发布了 HTML5 的工作草案。到了 2010 年，因为 HTML5 能解决非常实际的问题，所以在规范还未定稿的情况下，各大浏览器厂家就已经按捺不住了，开始对旗下产品进行升级以支持 HTML5 的新功能。这样，得益于浏览器的实验性反馈，HTML5 规范得到了持续完善，HTML5 以这种方式迅速融入对 Web 平台的实质性

改进中。虽然HTML5已经定稿，但HTML5仍在完善中，并计划于2016年底前发布HTML5.1推荐标准。HTML5 正式版的网址是：

```
http://www.w3.org/TR/2014/REC-html5-20141028/
```

HTML5 可以是存在于手机桌面的图标，也可以来自超级 App(如微信朋友圈)，以及搜索引擎、应用市场、广告联盟，为桌面和移动平台带来无缝衔接的丰富内容。HTML5 像血管一样连接一个个独立的 APP 器官并将血液输送到不同的 APP 中。

HTML5 的发展，有用户的需求在推动，也有技术开发者的需求在推动。学习 Objective-C 和 Java 比较费劲，既然会网页开发，为何不试试 HTML5 呢？它支持传统的桌面平台，又支持移动平台，包括 iOS、Android、BlackBerry、Windows Phone 等。这种跨平台的特性可以让所有平台共用一个 UI 代码库，节省几倍的开发时间。

1.1.2 谁在开发 HTML5

上面我们提到了一些开发 HTML5 的组织，下面进行详细介绍：

- **WHATWG：** 由来自 Apple、Mozilla、Google、Opera 等浏览器厂商的人组成，成立于 2004 年。WHATWG 开发 HTML 和 Web 应用 API，同时为各浏览器厂商以及其他有意向的组织提供开放式合作。
- **W3C：** W3C 下辖的 HTML 工作组目前负责发布 HTML5 规范。
- **IETF(Internet Engineering Task Force，互联网工程任务组)：** 这个任务组下辖 HTTP 等负责 Internet 协议的团队。HTML5 定义的一种新 API (WebSocket API)依赖于新的 WebSocket 协议，IETF 工作组正在开发这个协议。

1.1.3 HTML5 的主要新功能

- 语义化标记及新的特殊内容元素，比如 article、footer、header、nav、section
- 新的表单控件
- 用于绘画的 canvas 元素
- 用于媒介回放的 video 和 audio 元素
- 对本地离线存储的更好支持
- 地理位置
- 拖曳上传
- 多线程支持

1.2 HTML5 的优势

1.2.1 解决跨浏览器问题

一份互联网上对开发者的问卷调查表明：浏览器兼容性成了最大的问题，开发者对于浏

览器扩展规范能够统一、各大厂商浏览器兼容性能够提升的愿望最为迫切。在某个 Web 浏览器上可以正常运行的 HTML/CSS/JavaScript 等 Web 程序，在另一个 Web 浏览器上就运行不正常了。如果用一句话来描述这个问题的原因，可以说是“规范不统一”。规范未实现标准化，是造成这个问题的主要原因。

作为新生儿的 HTML5 在各种 Web 浏览器上的兼容性也一直是 Web 开发人员的心头大患。HTML5 的使命是详细分析各 Web 浏览器所具有的功能，然后以此为基础，要求这些浏览器的所有内部功能都要符合一个通用标准。比如，Google 分析了上百万个页面，从中分析出了 Div 标签的通用 ID 名称，发现其重复量很大，很多开发人员使用 id=header 来标记页眉区域，那为何不直接添加一个<header>标签呢？

如今各大浏览器厂商都支持 HTML5 标准，可以访问 https://html5test.com/查看自己的浏览器对 HTML5 的支持情况。那么以 HTML5 标准为基础来书写的程序在各浏览器上都能正常运行的可能性就大大提高了，这对于 Web 开发者和 Web 设计者都是一件令人可喜的事情。而且，今后开发者开发出来的 Web 功能只要符合通用标准，Web 浏览器也都是很愿意封装该功能的。

1.2.2　用户优先的原则

“用户即上帝”的宗旨也体现在 HTML5 中，用户怎么使用的，就怎么设计规范。HTML5 规范是基于用户优先准则编写的，这意味着在遇到无法解决的冲突时，规范会把用户放到第一位，其次是页面作者，再次是实现者(或浏览器)，接着是规范制定者(W3C/WHATWG)，最后才考虑理论的纯粹性。因此，HTML5 的绝大部分是实用的，只是有些情况下还不够完美。

例如，下面的代码在 HTML5 中是有效的：

```
<div id=wrapper>
<img SRC=logo.png aLt=logo>
```

我们不推荐这么使用 HTML5，HTML5 的适应性保证了用户体验，因为一旦由于开发人员的原因造成页面错误而导致不能正常显示，那么受折磨的肯定是最终用户。

```
<video>
<source src="movie.mp4">
<source src="movie.ogv">
<object data="movie.swf">
<a href="movie.mp4">download</a>
</object>
</video>
```

上面的代码包含了 4 个不同的层次：

- 如果浏览器支持video元素，也支持MP4，那么没什么好说的，用第一个视频。
- 如果浏览器支持video元素，支持ogv，那么用第二个视频。
- 如果浏览器不支持video元素，那么就要试试Flash影片了。
- 浏览器不支持video元素，也不支持Flash，还给出了下载链接。

HTML5 进行了优雅的降级，保证了最终用户的体验。

1.2.3　化繁为简的优势

HTML5 减少了对插件的依赖，HTML5 要的就是简单，避免不必要的复杂性，化繁为简无异于提供了更好的服务。HTML5 做了以下这些改进：

- 以浏览器原生能力替代复杂的 JavaScript 代码
- 新的简化的 doctype
- 新的简化的字符集声明
- 简单而强大的 HTML5 API

随后我们将详细讲解这些改进。为了实现所有的这些简化操作，HTML5 规范需要包括一系列定义明确的行为，任何歧义和含糊都可能延缓这一目标的实现。HTML5 规范已经变得非常大而且要求精确。

1.3　HTML5 基本语法

本书以人人微博网站为案例，在已有的基于 HTML4 的 Web 网页的基础上，修改为基于 HTML5 的 Web 网页。在学习项目案例之前，我们应该掌握 HTML4 和 CSS 的知识，熟悉 Tomcat 的部署和启动。项目需求如下：

人人微博是一种通过关注机制分享简短实时信息的广播式的社交网络平台，功能模块包括与用户相关的登录、注册、个人资料维护功能，与关系链相关的关注功能，与微博内容相关的发布微博、浏览微博功能。

- **用户登录：**用户可以使用手机号或邮箱登录微博，如图 1-1 所示。



图 1-1　微博首页

- **用户注册：**用户可以注册账号并设置昵称和性别，如图1-2所示。



图 1-2　用户注册

- **用户主页：**用户在个人主页上可以查看自己关注的好友，自己和好友发布的微博，并且可以实时发布微博，如图1-3所示。



图 1-3　用户主页

- **用户个人资料维护：**用户可以完善和修改自己的地址、头像、生日等个人信息，如图1-4所示。



图 1-4　微博个人信息

下面我们来运行案例代码：

(1) 请打开本书源代码 RrwbHtml4.rar。

(2) 将其解压到 Tomcat 目录的 Webapps 目录下，这里使用 Tomcat 默认的 8080 端口。

(3) 运行 Tomcat 目录的 bin 目录下的 startup.bat。

(4) 在浏览器的地址栏中输入 http://localhost:8080/rrwb/并回车，就能看到项目案例的主页了。

1.3.1　新的 DOCTYPE

还记得这段 HTML4 的 doctype 吗？

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

现在，HTML5 提供了一个很好的而且便于记忆的 doctype。

```
<!DOCTYPE html>
```

简化了吧，就这么多。我们只需要使用它，就可以告诉浏览器应处于标准兼容模式下。浏览器会根据 DOCTYPE 识别应该使用哪种显示模式，以及使用什么规则来验证页面。即使某个浏览器尚未实现 HTML5，页面仍然可以继续工作。

1.3.2　新的字符编码

在 HTML4 中，使用 meta 元素的形式指定文件中的字符编码，如下所示：

```
<meta http-equiv="content-type" content="text/html; charset=utf-8">
```

但是，在 HTML5 中，它会像 doctype 一样非常简洁：

```
<meta charset="utf-8">
```

请记住，我们的页面需要字符编码和 doctype 声明。

1.3.3　引用链接和样式表

因为链接和样式表将成为我们所看到或创建的常见元素，所以这里快速介绍一下如何将它们包含于网页中，这很重要。本书稍后对 CSS 和 JavaScript 有更多的介绍。

在 HTML4 中，使用 link 给网页添加样式，如下所示：

```
<link rel="stylesheet" type="text/css" href="styles/changeinfo.css"/>
```

HTML5 对此进行了简化，如下所示：

```
<link rel="stylesheet"  href="styles/changeinfo.css"/>
```

HTML5 向页面添加脚本则更加简单。在页面中添加一个<script>元素，然后添加 src 属性，指向需要使用的 JavaScript 文件的位置：

```
<script src="scripts/changeinfo.js"></script>
```

而 HTML4 添加脚本则需要指定 type 属性：

```
<script language="javascript" type="text/javascript" src="scripts/
    changeinfo.js"></script>
```

script 元素新增了 defer 属性和 async 属性，正常情况下浏览器会按页面顺序加载 js 文件和渲染，defer 属性表示当前 js 文件会在页面渲染后加载。async 属性表示异步(并行)加载和执行当前 js 文件。例如：

```
<!DOCTYPE HTML>
<html>
<head>
   <meta charset="utf-8">
   <title>HTML5</title>
   <!-- 大小为 1.88 KB -->
   <script defer onload="alert('loaded A')" src="http://cdnjs.cloudflare.com
       /ajax/libs/html5shiv/3.6/html5shiv.min.js" ></script>
   <!-- 大小为 250 KB -->
   <script async onload="alert('loaded B')" src="http://apps.bdimg.com
       /libs/jQuery/2.1.1/jQuery.js" ></script>
</head>
```

```
<body>
   Hello
</body>
</html>
```

运行的结果为：先进行页面显示，后显示弹出框，经过测试弹出“loaded A”和“loaded B”的顺序不确定。这两个属性都能够实现以非阻塞的方式加载脚本，区别在于执行的时机不同。

1.3.4　HTML5 标签的向后兼容性

如何让 IE6～IE8 浏览器支持 HTML5 元素呢？使用一小段代码就行，将它引入页面的 head 部分的后面，看下面的例子：

```
<!--[if lt IE9]>
<script
src="http://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.6/html5shiv.min.js">
</script>
<![endif]-->
```

HTML5 shiv 的主要作用就是让 IE 正常识别 HTML5 标签，就目前而言，只要知道需要引入它即可。如果不这么做，在 IE8 和更早期版本中就会得到不可预测的结果。我们可以查阅 Paul Irish 的文章“The History of the HTML5 Shiv”来了解这段简短而重要的脚本的历史，网址是：

```
http://paulirish.com/2011/the-history-of-the-html5-shiv/
```

此外还有构建在 HTML5 shiv 之上的另一个库，它被称为 Modernizr。为保证向后兼容性，它的核心部分包含了 HTML5 shiv。书中的案例会使用相对较简单的 HTML5 Shiv 来确保对于 IE8 和更早版本的兼容性。

现在我们来修改人人微博网站的页面的 doctype、字符编码、链接和样式表并加入 HTML5 shiv，修改后所有的页面都有类似于下面的代码：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>人人微博</title>
<link rel="stylesheet" href="styles/index.css" />
<script src="scripts/index.js"></script>
<script src="scripts/move.js"></script>
<!--[if lt IE9]>
<script
src="http://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.6/html5shiv.min.js">
</script>
<![endif]-->
</head>
</html>
```

1.3.5　<base>元素

为页面上的所有链接规定默认基准 URL。浏览器随后将不再使用当前文档的 URL，而是使用通过 base 元素指定的 URL 作为基准。例如：

```
<!DOCTYPE HTML>
<html>
<head>
   <meta charset="utf-8">
   <title>HTML5</title>
   <base href="https://www.baidu.com/" target="_blank"/>
</head>
<body>
    <a href="img/bdlogo.png">logo</a>
</body>
</html>
```

当点击 logo 链接时，则会打开网址 https://www.baidu.com/img/bdlogo.png。

完成了这部分的学习后，我们已经准备好了继续学习页面结构，下面让我们学习 HTML5 的文档结构和内容分组。

1.4　HTML5 内容分组

在 HTML5 之前，大部分网页制作方法都是利用 class 或 id 属性来命名网站的页面结构。比如：

```
<body>
    <div id="header">
        <div id="logo"></div>
        <div id="navigation">
            <ul>
                <li><a href="#">链接</a></li>
            </ul>
        </div>
    </div>
    <div id="content"></div>
    <div id="footer"></div>
</body>
```

而 HTML5 追加了几个与页眉、页脚、内容区块等文档结构相关联的结构元素，对于微博网站来说，导航菜单、微博正文、微博的评论等每一个部分都可称为内容区块，新增的结构元素可以对内容进行更精确的分组。运用 HTML5 可以这样写：

```
<body>
    <header>
        <div id="logo"></div>
```

```
            <nav>
                <ul>
                    <li><a href="#">链接</a></li>
                </ul>
            </nav>
        </header>
        <div id="content"></div>
        <footer></footer>
    </body>
```

HTML5 的新结构元素使得网页结构更加清晰、规范，使 HTML 更具语义化，语义化是指用合理的 HTML 标记及其特有的属性来格式化文档内容。通俗地讲，语义化就是对数据和信息进行处理，使得机器可以理解。这样做的好处有：

- **手持移动设备的无障碍阅读。**手持移动设备(如 PDA、智能手机等)可能对 CSS 的解析能力较弱，这时可能就需要更语义化的标签来体现页面的易读性。
- **盲人等一些障碍人士可以更好地阅读。**屏幕阅读器对不同标签所发出的声音是不同的，使用更语义化的标签能传达不同信息的重要性。
- **搜索引擎的友好收录。**虽然各大搜索引擎的排名规则不断地变化，但规则里的正则始终是要匹配标签的，如果能用更语义化的标签，搜索引擎就能够根据标签的使用来确定各关键字的权重。

1.4.1 <header>元素

<header>元素可以用作网站的“题头”区域，也可以作为对其他内容(如<article>元素)的简要介绍。在稍后介绍的<article>元素中可以拥有它自己的题头，用来包含有关作者、发布日期以及文章标题等信息。

1.4.2 <footer>元素

和<header>元素一样，它也可以视情况在同一个页面上多次出现。例如微博网站的页脚可以用它，我们在<footer>中加入团队简介、联系方式、版权信息等。

1.4.3 <address>元素

<address>元素不只是用来呈现电子邮箱或真实地址，还用于标记<article>元素或整个文档的联络信息。还可用来展示与文档相关的联系人的所有联系信息。

1.4.4 <article>元素

<article>元素是一个独立的内容块，它既可以独立存在，也可以被重用。可以想象一些常见的例子，比如一篇微博、一个论坛帖子、一段时尚评论、一段新闻稿或一个交互式小工具。最基本的经验是，如果一段内容可以被“聚合”，或者可以独立于网站其余环境进行分享，就应该使用<article>元素进行标记。比如一篇新闻：

```
<article>
```

```
    <h1>Safari 5 released</h1>
    <p>7 Jun 2010. Just after the announcement of the new iPhone 4 at WWDC,
    Apple announced the release of Safari 5 for Windows and Mac......</p>
</article>
```

1.4.5　<nav>元素

<nav>元素最常用的用途是作为网站的主导航。很多情况下，开发者都使用无序列表编写导航，它用于链接到网站内的其他页面，或是链接到一个页面的其他部分。HTML5 的编写方式为：

```
<nav class="header_Llist">
<ul>
<li><a href="#">首页</a></li>
<li><a href="guangchang.html" target="_blank">广场</a></li>
<li><a href="#">微群</a></li>
<li><a href="#">应用</a></li>
<li><a href="#">游戏</a></li>
</ul>
</nav>
```

1.4.6　<section>元素

<section>元素定义文档中的节(区段)，比如章节、页眉、页脚或文档中的其他部分。<section>元素的使用方式和<div>元素的使用方式相似。不过，与<div>元素不一样的是，<section>元素具有语义含义，它是一组相关内容的组合。<section>元素中可以包含<article>元素，<article>元素也可以包含<section>元素，<article>元素是一个特殊的 section，它比 section 具有更明确的语义，它代表一个独立的、完整的相关内容块。一般来说，<article>元素会有标题部分(通常包含在<header>内)，有时也会包含<footer>元素。

```
<article>
    <hgroup>
    <h1>苹果</h1>
    <h2>品尝美味的苹果!</h2>
    </hgroup>
    <p>苹果的品种有很多。</p>
    <section>
    <h1>红苹果、芳香</h1>
    <p>这种红苹果在超市里非常常见。</p>
    </section>
    <section>
    <h1>绿苹果、较酸</h1>
    <p>绿苹果可以用来制作果汁。</p>
    </section>
</article>
```

1.4.7　<hr>元素

<hr>元素用于在页面中创建一条横跨页面的横线，分隔页面中不适于使用新标题的独立区域。HTML5 中不再支持 align、noshade、size 以及 width 属性。

1.4.8　<blockquote>元素

<blockquote>与</blockquote>之间的所有文本都会从常规文本中分离出来，经常会在左右两边进行缩进(增加外边距)，而且有时会使用斜体，还可以使用 cite 属性指明引用内容的来源的 HTTP 地址。

1.4.9　<aside>元素

<aside>元素用来表示当前页面或文章的附属信息部分，它可以包含与当前页面或主要内容相关的引用、侧边栏、广告、导航条，以及其他类似的有别于主要内容的部分。<aside>元素主要有以下两种使用方法：

- 被包含在<article>元素中作为主要内容的附属信息部分，其中的内容可以是与当前文章有关的相关资料、名词解释等。
- 在<article>元素之外使用作为页面或站点全局的附属信息部分。最典型的是侧边栏，其中的内容可以是友情链接、博客中的其他文章列表、广告单元等。

1.4.10　<hgroup>元素

<hgroup>元素用于组织具有一些逻辑联系的多级标题，例如子标题、标题信息、可选标题等内容，可以被包含在<section>、<article>等元素中。

1.4.11　<a>元素

HTML5 中的<a>元素可以包含段落、列表、表格等块级元素。下面的例子演示的整个广告块被包含在<a>元素中：

```
<aside class="advertising">
 <h1>广告</h1>
 <a href="">
  <section>
   <h1>岁末清仓，省时、省力!</h1>
   <p>热卖!</p>
   <p>只要 9.9 元</p>
  </section>
 </a>
 <a href="">
  <section>
   <h1>团购年货</h1>
   <p>1 元起</p>
   <p>Go!</p>
  </section>
 </a>
</aside>
```

1.4.12　HTML5 的文档大纲

就像 Word 中的大纲视图，HTML 也有自己的大纲。比如，新闻网站中文章的结构与报纸上文章的结构类似。每一篇文章包含标题、文本段落以及一些图片(有时可能以视频代替图片)。相同点非常明显，而唯一的不同就是，在报纸上可以在一个版面上容纳多篇故事，而在网页上每篇故事则倾向于独占一个页面。HTML4 的大纲是由<h1>、<h2>、<h3>、<h4>、<h5>、<h6>完成的，由于整个 HTML 文档是一个独立的元素，因此应该只拥有一个总体摘要，在摘要树的顶端是唯一的<h1>元素。要了解 HTML5 的文档大纲，需要知道主体结构元素和非主体结构元素的区别以及 HTML5 的新大纲算法。

1. 结构元素和非结构元素的区别

上面学习了 HTML5 的新元素，新增的主体结构元素包括<article>元素、<section>元素、<nav>元素及<aside>元素。主体结构元素能够表示内容区块，例如对于书籍来说，章、节都可以称为内容区块。对于微博网站来说，导航菜单、微博正文、微博的评论等每一个部分都可称为内容区块。新增的非主体结构元素包括<header>元素、<hgroup>元素、<footer>元素及<address>元素，非主体结构元素表示逻辑结构或附加信息。

2. HTML5 的新大纲算法

由于加入新的内容分组元素，HTML5 的文档结构比大量使用<div>元素的 HTML4 的文档结构清晰、明确了很多。在微博网站中，会有多篇包含于同一个 HTML 文档中的独立文章。这些文章都可以拥有自己的逻辑摘要，并独立于页面的其他部分。基于这个原因，HTML5 引入了一种能力，即可以在同一页面中加入多个<h1>元素，并将其用于页面中不同的分区元素内。大纲算法允许用户代理从一个 Web 页面生成一个信息结构目录，让用户对页面有一个快速概览。类似书籍、PDF、帮助文档等，都有一个清晰的目录结构，用户能方便地定位所需内容。一个结构良好的大纲，不仅是对搜索引擎的优化，更是为借助于屏幕阅读器浏览网页的失明人士或弱视力用户提供了巨大的帮助。如果想要查看这段代码的大纲，可以试试 Geoffrey Sneddon 做的大纲工具：

```
https://gsnedders.html5.org/outliner/
```

我们来验证如下 HTML5 代码：

```
<div>
  <h6>名著推荐</h6>
  <section>
    <h1>国内名著</h1>
    <article>
      <h1>红楼梦</h1>
      <p>红楼梦讲的是..</p>
    </article>
    <article>
      <h3>边城</h3>
```

```
    <p>边城讲的是..</p>
  </article>
 </section>
 <section>
  <h5>国外名著</h5>
  <article>
   <h6>战争与和平</h6>
   <p>战争与和平讲的是..</p>
  </article>
  <article>
   <h1>红与黑</h1>
   <p>红与黑讲的是..</p>
  </article>
 </section>
<div>
```

生成的文档大纲如图 1-5 所示。

```
1. 名著推荐
  1. 国内名著
    1. 红楼梦
    2. 边城
2. 国外名著
    1. 战争与和平
    2. 红与黑
```

图 1-5　生成的文档大纲

为什么 h1 到 h6 的层级在这里没有表现出来？原因是此时大纲是由节点元素生成的，而非标题元素。HTML5 的新标签 section、article、aside、nav 会生成显性节点(explicit sections)，每个显性节点内部又有自己的标题结构。这也就是为什么 HTML5 允许多个 h1 存在的原因。不过，在全部浏览器、屏幕阅读器都完美支持 HTML5 之前，建议还是同时考虑标题结构，优雅降级。上面的结构可以改成这样：

```
<div>
 <h1>名著推荐</h1>
 <section>
  <h2>国内名著</h2>
  <article>
   <h3>红楼梦</h3>
   <p>红楼梦讲的是..</p>
  </article>
  <article>
   <h3>边城</h3>
   <p>边城讲的是..</p>
  </article>
 </section>
 <section>
```

```
    <h2>国外名著</h2>
    <article>
      <h3>战争与和平</h3>
      <p>战争与和平讲的是..</p>
    </article>
    <article>
      <h3>红与黑</h3>
      <p>红与黑讲的是..</p>
    </article>
  </section>
<div>
```

我们可以得出结论：显性节点(section 等)可以包含隐性节点(h1～h6)，反之则不可以。有时会出现未命名节点(untitled sections)，比如：

```
<nav class="header_Llist">
    <ul>
        <li><a href="#">首页</a></li>
            <li><a href="guangchang.html" target="_blank">广场</a></li>
              <li><a href="#">微群</a></li>
              <li><a href="#">应用</a></li>
              <li><a href="#">游戏</a></li>
     </ul>
</nav>
<footer>
  <h3>我是 footer 可是为什么我成为了文档标题啊 </h3>
</footer>
```

生成的文档大纲如图 1-6 所示。

1. 我是footer可是为什么我成为了文档标题啊
   1. *Untitled Section*

图 1-6　生成的文档大纲

因为 nav 里并没有包含任何标题元素，<nav>元素会生成一个 untitled section，这不是错误，也不会被认为是不好的 HTML5 结构。但是 section、article 还是建议给予适当的标题。为什么<footer>元素的标题成了文档标题呢？因为文档标题是文档中第一个非节点元素里的第一个标题元素。

1) 实现微博首页

掌握了 HTML5 的文档结构、结构元素之后，就可以用这些基础知识来搭建一个语义清晰、结构分明的 HTML5 网站了。下面介绍如何利用 HTML5 的各种结构元素搭建微博首页。微博首页的主体结构如图 1-7 所示。

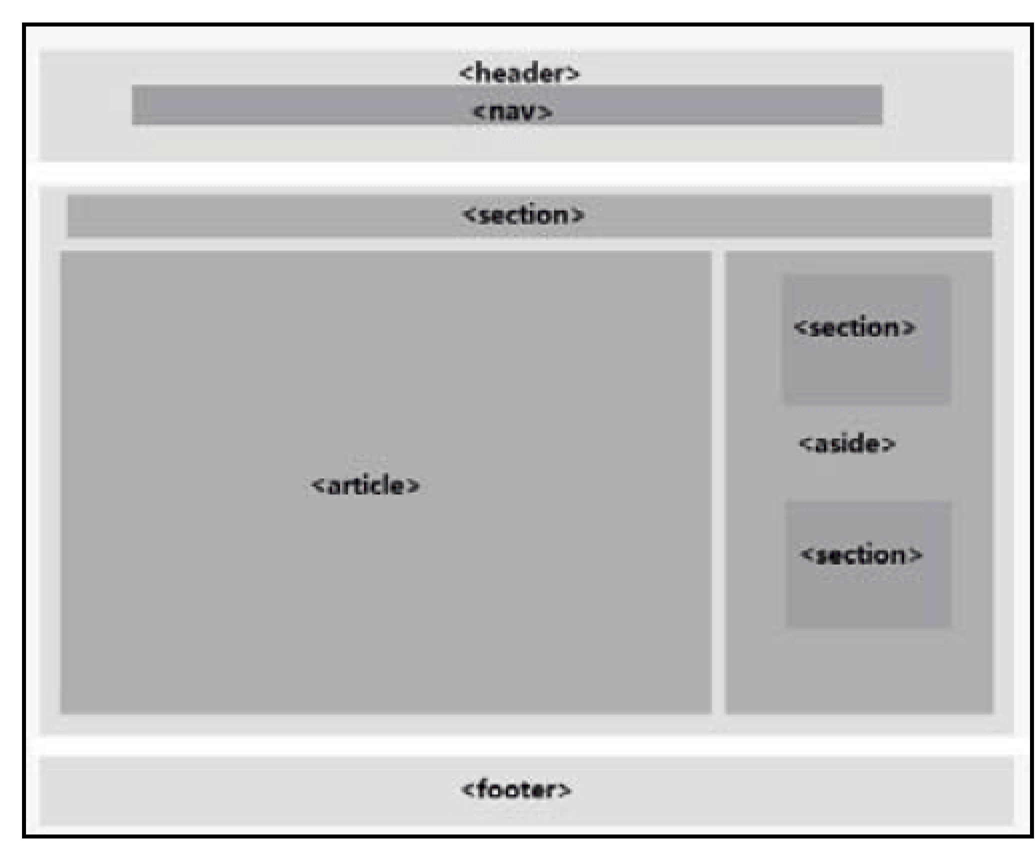


图 1-7　微博首页的主体结构

该页面主要分为 4 个部分：第 1 部分为网页头部，显示该微博网站的网站标题、网站导航链接、搜索框；第 2 部分为网页宣传栏和登录框，显示醒目的内容；第 3 部分由微博文章、达人推荐、微博应用组成，是该微博网站首页上的主要内容；第 4 部分为页面底部的版权信息和地址信息。

(1) 前面介绍过，<header>元素是一种具有引导和导航作用的结构元素，通常需要放置一个标题。在首页上，一般将标题与整个网站的导航链接作为整体网页的标题放置在<header>元素中。如前所述，<nav>元素是一个可以作为页面导航的链接组，其中的导航元素链接到其他页面或当前页面的其他部分。这里我们使用图片 logo 作为标题，并把整个网站的导航链接放在该<nav>元素中。该部分的结构代码如下所示：

```
<header id="global_header">
       <div class="header">
            <div class="header_logo"><h1><img src="images/logo_update_02.png"
                 width="190" height="34" alt="人人微博"/> </h1> </div>
             <nav class="header_Llist">
             </nav>
    </div>
</header>
```

我们使用了一个小技巧，可以指定<img>元素的 alt 属性，并使用<h1>元素包含<img>元素，这样 alt 的属性就会作为标题显示在文档大纲中，由于<h1>元素的默认边距太宽会使得<img>元素显示在外部，我们在 index.css 中添加如下代码：

```
h1,h2,h3,h4,h5,h6{padding:0; margin:0}
```

(2) 将网页宣传栏和登录框修改为<section>元素：

```
<section class="banner">
    <div id="show"><img src="images/banner2.jpg" alt="图片" /></div>
    <div id="msg"> </div>
     <div id="usrlogin"> </div>
 </section>
```

(3) 实现微博网站首页的主要内容部分，左侧为微博文章列表，每一篇文章都可以包含

自己独立的内容，我们使用<article>元素。右侧是侧边栏，可以将微博达人、下载微博应用的链接等不是主体内容的其他附属内容放在<aside>元素中。由于微博达人、下载微博应用是不同的内容，可以使用<section>元素表示不同的节、不同的内容。

代码结构如下：

```
<div id="content">
<div id="saying">
<div id="sayframe">
<article id="weibo1" class="weibo"></article>
<article id="weibo2" class="weibo"></article>
<article id="weibo3" class="weibo"></article>
</div>
</div>
</div>
<aside>
<section class="fav_people">  </section>
<section> </section>
</aside>
```

(4) 实现页面底部信息，包含版权信息和联系信息，这里使用<address>元素完成联系信息。代码如下：

```
<footer id="footer">
Copyright 人人微博 All rights reserved.
<address>联系我们(QQ):12345678</address>
</footer>
```

至此，整个首页的改版就完成了，整个页面看起来和原来没什么区别，但是它已经是全新的 HTML5 页面了，也可以通过大纲工具查看微博首页的文档大纲。

2) 实现微博个人主页

我们再来修改微博个人主页为 HTML5 版本。该页面主要分为 4 个部分：第 1 部分为网页头部，显示该微博网站的网站标题、网站导航链接、搜索框，与微博首页是相同的；第 2 部分是左边栏，包括我的信息、每日记事、我的关注；第 3 部分包括发布微博、最新微博列表；第 4 部分为页面底部的版权信息和地址信息，与微博首页是相同的。

(1) 按照 index.html 头部代码修改微博个人主页的头部。

(2) 在左边栏中，可以用<aside>元素来显示。将我的信息、每日记事、我的关注放在侧边栏中，结构代码如下所示：

```
<aside class="cont_Left">
     <section class="cont_Leftone">
        <div class="cont_threetop">
        <h3><span class="cont_tretop1 cont_two_n">我的信息</span></h3>
       <div>
    </section>
    <section class="cont_Lefttwo">
        <div class="cont_threetop">
```

```
        <h3><span class="cont_tretop1 cont_two_n">每日记事</span></h3>
       <div>
    </section>
   <section class="cont_Leftthree">
        <div class="cont_threetop">
        <h3><span class="cont_tretop1 cont_two_n">我的关注</span></h3>
       <div>
    </section>
</aside>
```

(3) 第 3 部分微博列表使用<article>元素来显示每一篇微博文章。下面给出部分代码：

```
 <article class="cont_down4_b">
   <span class="fo">张三：</span>
   世界上最美丽的天坑，阿曼 Bimmah 天坑，当地人则说，它是月亮上掉下来的一块碎片砸进地里
   造成的。
   [图片]<img src="images/4c1d5964jw1dun6cuiqtrj.jpg" />
 </article>
```

(4) 按照 index.html 底部代码修改微博个人主页的底部。

1.5　HTML5 使用列表

1.5.1　使用<ul>元素创建无序列表

我们经常在网页上看到不同的文字列表和图片列表，比如图 1-8 所示的无序列表。



图 1-8　无序列表

这是使用<ul>(即“unordered list”，无序列表之意)元素创建的。在 HTML4 中，<ul>元素的 compact 和 type 属性是不推荐使用的。在 HTML5 中，不再支持这两个属性。代码如下：

```
<ul>
  <li><a href="#">热搜榜首页</a></li>
  <li><a href="#">实时热搜榜</a></li>
  <li><a href="#">综合热搜榜</a></li>
  <li><a href="#">时事热搜榜</a></li>
  <li><a href="#">影视热搜榜</a></li>
```

```
  <li><a href="#">财经热搜榜</a></li>
  <li><a href="#">名人热搜榜</a></li>
</ul>
```

1.5.2　使用<ol>元素创建有序列表

有时候我们希望给列表加上序号，以表示有序的列表，只需要将每一个列表项嵌套于<ol>元素内。比如下面的例子：

```
<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

在 HTML4 中不赞成使用 start 和 type 属性，但是在 HTML5 中支持。我们来看下面的例子：

```
<ol start="50">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

运行效果如图 1-9 所示。

HTML5 列表

50. Coffee
51. Tea
52. Milk

图 1-9　有序列表的运行效果

type 属性让我们可以使用数字(1、2、3)、字母(A、B、C)或罗马数字(i、ii、iii)标识序号。表 1-1 给出了可用选项。

表 1-1　textarea 元素属性

属 性 值	类　型	描　述
1	decimal	小数(默认)
a	lower-alpha	小写拉丁字母
A	upper-alpha	大写拉丁字母
i	lower-roman	小写罗马数字
I	upper-roman	大写罗马数字

HTML5 中新增了 reverse 属性，让我们可以反转有序列表的顺序。例如：

```
<ol start="4" reversed type="A">
  <li>Coffee</li>
```

```
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

运行效果如图 1-10 所示。

HTML5 列表

D. Coffee
C. Tea
B. Milk

图 1-10　有序列表反转的运行效果

实现导航

我们可以使用列表实现顶部的导航栏，代码如下：

```
<nav class="header_Llist">
    <ul>
        <li><a href="index.html" target="_self">首页</a></li>
        <li><a href="guangchang.html" target="_self">广场</a></li>
        <li><a href="game.html" target="_blank">游戏</a></li>
    </ul>
</nav>
```

1.6　HTML5 语义元素

1.6.1　<ruby>元素

<ruby>元素用来显示包含文字的注音。ruby 原本是印刷语言，指放于表意文字上方或右边的拼音或批注，广泛应用于日文和中文。<ruby>元素内可以包含<rt>元素，<rt>元素用来显示要加上注音标注的文字，也就是用来显示标注的注音符号，<rt>元素的内容则可放置<ruby>元素外的任何内联元素。例如：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
</head>
<body>
    <ruby>
        前端开发  <rt>qian duan kai fa</rt>
    </ruby>
</body>
</html>
```

运行效果如图 1-11 所示。

qian duan kai fa
前 端 开 发

图 1-11　<ruby>元素的运行效果

1.6.2　<small>元素

<small>元素代表小打印体，如版权信息、条款或法律信息。缩小字体应该由 CSS 来完成。

```
<small>Copyright 人人微博 All rights reserved.</small>
```

1.6.3　<time>元素

<time>元素表示内容是日期类型，可以是各种格式的时间、时区偏移字符串、持续时间字符串，它有 datetime 属性，用来表示以计算机可识别的格式记录相同的日期和时间。例如：

```
<p> 会议开始于<time>20:00</time>./p>
```

如省略 datetime 属性，则<time>元素的内容必须符合 datetime 的有效格式，如表 1-2 所示。

表 1-2　time 的有效格式

格　式	示　例
有效的日期字符串	<time>2015-1-1</time>
有效的“月”日期字符串	<time>2015-1</time>
有效的“月日”日期字符串	<time>1-1</time>
有效的本地日期和时间字符串	<time>2015-1-1T11:11</time>
有效的时间字符串	<time>11:11</time>
有效的时区偏移字符串	<time>+0000</time> <time>-0800</time>
有效的全球日期和时间字符串	<time>2015-1-1T14:54+0000</time> <time>2015-1-1T06:54-0800</time> <time>2015-1-1T06:54:39-0800</time>
有效的周字符串	<time>2015-W1</time>
有效的非负整数年份字符串	<time>2015</time>
有效的持续时间字符串	<time>2h 3m 38s</time>
有效的日期字符串	<time>2015-1-1</time>

下面的例子以人类可读的形式展现有效日期并提供计算机可以识别的日期格式：

```
<p>我们见面的日期是 <time datetime="2009-09-23">星期二</time>.</p>
<time datetime="2015-1-1">New Year's Day 2013</time>
```

1.6.4　<details>元素

<details>元素定义元素细节，用户可以查看，也可以通过点击进行隐藏。请看下面的例子：

```
<details>
<summary>HTML5</summary>
<p>用户优先</p>
<p>化繁为简</p>
<p>新的 DOCTYPE</p>
</details>
```

<summary>元素从属于<details>元素，用鼠标点击<summary>元素中的内容文字时，<details>元素中的其他所有从属元素将展开或收缩。<details>元素的 open 属性定义了 details 是否可见，添加该属性后，在画面打开时细节区域则会处于展开状态。上述代码的运行效果如图 1-12 所示。



图 1-12　<details>元素的运行效果

1.6.5　<progress>元素

<progress>元素表示进度条，它的属性包括 value 和 max，value 表示当前进度，max 表示需要完成的进度。value 属性和 max 属性只能指定为有效的浮点数。value 属性的值必须大于 0，且小于等于 max 属性。例如：

```
<progress value="70" max="100">70 %</progress>
```

运行效果如图 1-13 所示。



图 1-13　<progress>元素的运行效果

1.6.6　<datalist>元素

<datalist>元素定义选项列表。datalist 及其选项不会被显示出来，它仅是合法的输入值列表。比如：

```
<input list="cars" />
<datalist id="cars">
    <option value="BMW"/>
    <option value="Ford"/>
```

```
    <option value="Volvo"/>
</datalist>
```

运行效果如图 1-14 所示。

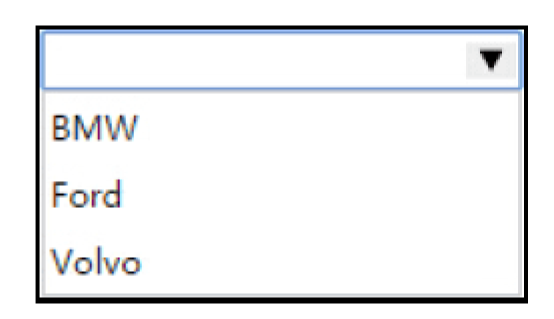


图 1-14　<datalist>元素的运行效果

1.6.7　<meter>元素

<meter>元素表示已知范围的度量衡，比如磁盘用量、调查结果等。常用属性有：

- **max：** 规定范围的最大值，默认是 1。
- **min：** 规定范围的最小值，默认是 0。
- **high：** 规定被视为高的值的范围。
- **low：** 规定被视为低的值的范围。
- **value：** 必需的属性，规定度量的当前值。

比如下面的例子：

```
<p><meter high=".8" value=".5"></meter></p>
<p>如果 value 的值大于 max 的值，value 的值将变为 max 的值<meter value="5"></meter></p>
<p>如果 value 的值大于 high 的值，以黄色警告显示值的度量<meter value=".9" high=".8">
  </meter></p>
```

这段代码的运行效果如图 1-15 所示。

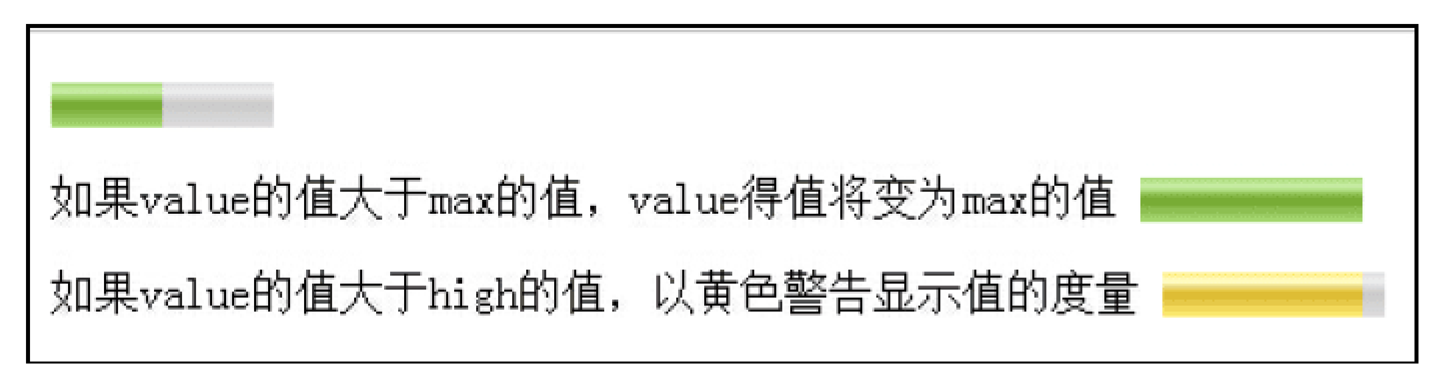


图 1-15　<meter>元素的运行效果

1.6.8　<mark>元素

<mark>元素表示带记号的文本(高亮显示文本)，其初衷是将注意力吸引到文档中标记的部分，类似于纸质书中使用高亮笔标注内容。比如：

```
<p>我们的任务是熟悉<mark>前端开发!</mark></p>
```

这段代码的运行效果如图 1-16 所示。

我们的任务是熟悉前端开发!

图 1-16　<mark>元素的运行效果

1.6.9　<figure>及<figcaption>元素

<figure>元素用来表示网页上一块独立的内容，将其从网页上移除后不会对网页上的其他内容产生任何影响。可包含插图、统计图、代码块等。它包含一个可选的<figcaption>元素，用来定义<figure>元素的标题，<figcaption>元素应该被置于<figure>元素的第一个或最后一个子元素的位置，<figure>元素内最多只允许放置一个<figcaption>元素。比如：

```
<p> 泰山日出是岱顶奇观之一，也是泰山的重要标志</p>
<figure>
  <figcaption>泰山日出</figcaption>
  <img src="richu.jpg"/>
</figure>
```

这段代码的运行效果如图 1-17 所示。



图 1-17　<figure>及<figcaption>元素的运行效果

1.7　HTML5 多媒体

HTML5 为网页带来了全新的标准原生视频和音频渲染规范，并增加了<audio>和<video>元素。<audio>和<video>元素的出现让 HTML5 的媒体应用多了新选择，开发人员不必使用插件就能播放音频和视频。本章先介绍一些<audio>和<video>元素的基础使用方法，然后考虑一些常见的编码和跨浏览器问题。

1.7.1　<audio>元素

<audio>元素也是 HTML5 的新元素，允许开发人员在其网站上嵌入音乐，下面是一个<audio>示例：

```
<audio src="/test/audio.ogg">
<p>你的浏览器不支持 audio 元素.</p>
</audio>
```

万一某些用户的浏览器不支持<audio>元素，那么他们会得到一个提示。还可以将音频源文件的位置放在元素之间：

```
<audio>
<source src="music.ogg">
</audio>
```

下面我们介绍<audio>元素的属性。HTML5 中的<audio>元素和<video>元素有很多相同之处，两者都支持的操作有播放、暂停、静音/消除静音、加载等，因此通用的动作形成了<audio>元素和<video>元素的通用属性。表 1-3 给出了通用媒体属性。

表 1-3　通用媒体属性

属　性	解　释
autoplay	将媒体文件设置为创建后自动播放，或者查询是否已设置为 autoplay
loop	如果媒体文件播放完毕后能重新播放则返回 true，或者将媒体文件设置为循环播放(或不循环播放)
currentTime	以秒为单位返回从开始播放到现在所用的时间。在播放过程中，设置 currentTime 来进行搜索，并定位到媒体文件的特定位置
controls	显示或隐藏用户控制界面，或者查询用户控制界面当前是否可见
volume	在 0.0 到 1.0 之间设置音量的相对值，或者查询当前音量的相对值
muted	为音频文件设置静音或消除静音，或者检测当前是否为静音
autobuffer	通知播放器在媒体文件开始播放前，是否进行缓冲加载。如果媒体文件已设置为 autoplay，则忽略此特性
preload	如果出现该属性，则视频在页面加载时进行加载，并预备播放。如果媒体文件已设置为 autoplay，则忽略此特性

此外，还有一些只读属性，比如 duration、paused、ended、startTime 等。

1.7.2　<video>元素

先用一段简单的代码来显示视频：

```
<video src="http://v2v.cc/~j/theora_testsuite/320x240.ogg" controls autoplay>
  Your browser does not support the <code>video</code> element.
</video>
```

这里声明了 autoplay 属性，这样一来，页面加载完成后，视频马上会被自动播放。运行效果如图 1-18 所示。

<video>元素还有额外的属性，如表 1-4 所示。

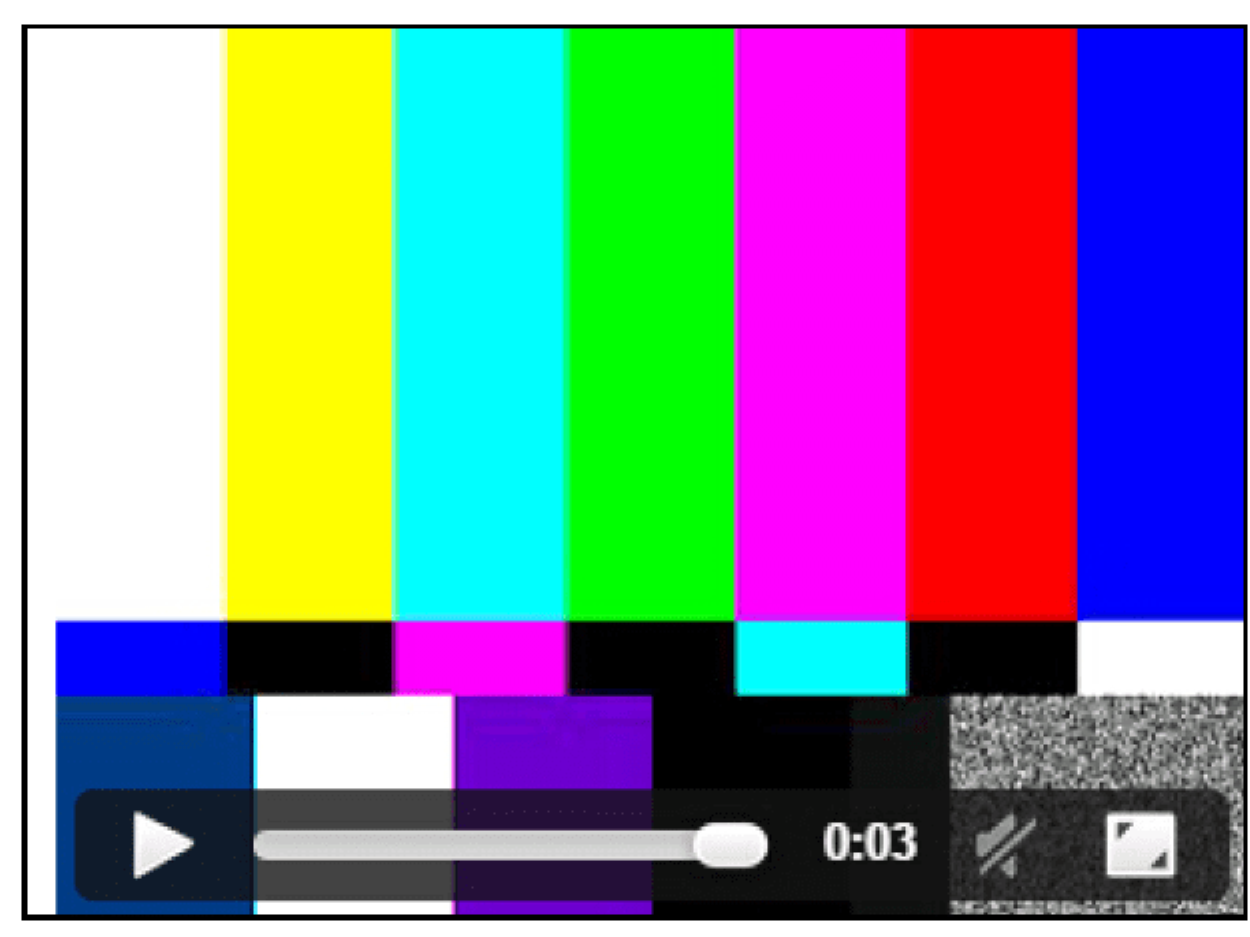


图 1-18　<video>元素的运行效果

表 1-4　<video>元素的属性

属　性	描　述
poster	在视频加载完成之前，代表视频内容的图片的 URL 地址。该特性不仅可读，而且可以修改，以便更换图片
width height	读取或设置显示尺寸。如果设置的宽度与视频本身的大小不匹配，则可能导致居中显示或出现黑色条状区域
videoWidth videoHeight	只读，返回视频的固有或自适应的宽度和高度

1.7.3　音频、视频容器和编解码器

HTML5 遇到的最大挑战是视频编码问题，这需要了解音频、视频容器和编解码器。

主流视频容器支持如下视频格式：

- Audio Video Interleave(.avi)
- Flash video(.flv)
- MPEG 4(.mp4)
- Matroska(.mkv)
- Ogg(.ogv)

不论是音频文件还是视频文件，实际上都只是容器文件，这类似于压缩了一组文件的 ZIP 文件。视频文件(视频容器)包含音频轨道、视频轨道和其他一些元数据。视频播放时，音频轨道和视频轨道是绑定在一起的。元数据部分包含该视频的封面、标题、子标题、字幕等相关信息。

下面是一些音频编解码器：

- AAC
- MPEG-3
- Ogg Vorbis

下面是一些视频编解码器：

- MPEG-4(H.264)
- VP8(WebM)
- Ogg Theora

主要的挑战是 MPEG-4 编码，它要求任何使用该产品的厂商购买授权许可，Apple、Microsoft 支持具有专利保护的 MPET-4 格式，Google、Opera、Mozilla 支持免费、开源或者诸如 WebM 或 OGG/Theora 等版权免费的视频格式。没有任何一种编解码器可以被所有浏览器厂商接受并在其产品中提供支持，如图 1-19 所示。

BROWSER	H264	H265	VP8	VP9	AAC	MP3	VORBIS	OPUS
Chrome for Desktop	30+	-	30+	30+	30+	30+	30+	33+
Internet Explorer for Windows	9+	-	-	-	9+	9+	-	-
Firefox for Desktop	22+ [1]	-	20+	28+	22+ [1]	22+ [1]	20+	20+
Safari for Mac	3+	-	-	-	3+	3+	-	-
Opera for Desktop	-	-	11+	16+	-	-	11+	12+
Safari for iOS	3+	-	-	-	3+	3+	-	-
Stock Browser for Android	2.3+	-	4.0+ [2]	-	2.3+	2.3+	4.0+ [3]	-
Chrome for Android	30+	- [4]	30+ [2]	-	30+	30+	30+	-
Internet Explorer for WinPhone	7.5+	-	-	-	7.5+	7.5+	-	-
Firefox for Android	20+	-	20+ [2]	28+ [2]	20+	-	20+	20+

1. Firefox supports H264/AAC/MP3 on OSX as of version 35.
2. Software decoding only, impacting performance and battery life.
3. *canPlayType()* not reporting on Vorbis, but audio is playing
4. Android Lollipop contains a software decoder that makes H265 play in Chrome.

图 1-19　浏览器与视频编码

HTML5 规范本来打算指定编解码器的计划实施起来困难重重，最终还是放弃了对编解码器的要求，并支持主要的三种格式：MP4、WebM、Ogg/Ogv。我们要做的只能是熟悉各种浏览器的支持情况，针对不同的浏览器环境对媒体文件进行重新编码。这里推荐使用 mirovideoconverter，这是一款非常易用且完全免费的软件，它支持多种不同视频类型之间的转换，地址为 http://www.mirovideoconverter.com/。

我们可以在 type 属性中设置多媒体类型和多媒体的编码，比如：

```
<source src="foo.mp4" type="video/mp4; codecs=avc1.42E01E, mp4a.40.2">
<source src="foo.ogg" type="video/ogg; codecs=dirac, speex">
```

如果我们明确知道浏览器支持某种类型，那么明确写清楚这种类型显然更明智。否则，最好是省略 type 属性，让浏览器自己检测编码。因为如果 type 属性中指定的类型与源文件不匹配，浏览器可能就会拒绝播放。

1.7.4　跨浏览器问题

我们肯定希望所有主流浏览器都能够播放音频或视频，我们可以准备多个视频和音频源，浏览器可以从这么多的来源中自动选择，下面通过使用多个<source>元素，在<video>元素中提供多种格式的视频文件。代码如下：

```
<video controls>
  <source src="somevideo.webm" type="video/webm">
  <source src="somevideo.mp4" type="video/mp4">
  <source src="somevideo.ogv" type="video/ogg">
</video>
```

对于不支持 HTML5 多媒体元素的旧浏览器，可以添加额外的向后兼容内容，我们可以使用 Flash 播放器提供音频或视频，比如：

```
<video src="video.ogv" controls>
    <object data="flvplayer.swf" type="application/x-shockwave-flash">
      <param value="flvplayer.swf" name="movie"/>
    </object>
</video>
```

下面我们在首页实现视频微博的功能，使用<video>元素在首页播放一段视频，代码如下：

```
<section id="play_weibo">
    <h3>视频微博</h3>
    <div id="pic_weibo">
       <video width="320" height="240" controls poster="images/pic_weibo.png">
           <source src="video/big.ogv" type="video/ogg">
           <source src="video/big.mp4" type="video/mp4">
       </video>
    </div>
    <div id="mobile"><img src="images/mobile.png"/></div>
</section>
```

1.8　HTML5 表单介绍

HTML5 表单并没有完全修改以前的表单规范，HTML5 表单规范更加注重对现有的 HTML 表单功能进行改进，以使其包含更多控件类型。HTML5 表单仍然使用<form>元素作为容器，我们可以在其中设置基本的提交特性。之前所讲的表单控件，如文本框、单选按钮、复选框等，都兼容原来的使用方法。当用户提交页面时，表单仍然用于向服务器发送表单中控件的值，我们仍可放心地使用 JavaScript 操作表单控件或者编写自己的处理函数。

<form>元素除了带有 action、method、target 属性外，还可以带有以下属性：novalidate、autocomplete、enctype。

novalidate 属性是一个布尔值，用来指定表单在提交时是否应该进行校验。如果该属性存

在，则浏览器不会在提交表单前校验表单。例如：

```
<form action="example.jsp"  novalidate>
```

autocomplete 属性指明浏览器是否应该自动填写表单值。将之设置为 off 指明浏览器不应该自动填写任何内容，默认值为 on。例如：

```
<form action="example.jsp" autocomplete="off" >
```

enctype 属性用于指定浏览器在将数据发送到服务器之前如何对其进行编码，浏览器通常支持三种类型的编码方式：application/x-www-form-urlencoded 作为默认值是大多数表单使用的标准方法，因为一些字符，如空格、加号以及一些非字母数字字符不能直接发送到服务器，在使用此属性值时，这些字符会被其他可用的字符替代；multipart/form-data 属性值允许数据分成几个部分传送，每一个连续部分以其在表单中出现的顺序对应一个表单控件，该方式通常被用于用户需要向服务器上传文件、图片时，每一个部分可以有一个属于它自己的可选的 content-type 头部信息，用来指定所属表单控件的数据类型；text/plain 以纯文本形式不加修改地将数据发送至服务器。

1.9　HTML5 新增的输入元素

在继续学习之前，我们看一下在人人微博网站中的哪些地方用到了表单？有注册表单、登录表单、个人信息表单、微博发布表单。可见人人微博的核心功能都是通过表单实现的，表单的创建不仅要符合用户的操作需求和业务需求，还要考虑与服务器端数据的交互。以个人信息页面(如图 1-20 所示)为例，用户的有些需求使用 HTML4 开发会很费劲而且是重复性的工作，而 HTML5 表单可以利用新的输入元素和属性满足这些需求。



图 1-20　HTML4 个人信息页面

本节将介绍新的 HTML5 输入类型以及一些新属性，它们极大地提升了<input>元素的输入体验，特别是在移动设备上，使得用户更容易填写网页表单。旧浏览器会将任何未知类型的<input>元素按 text 类型对待。这意味着即使新输入元素和新属性不可用，我们也仍然能够收集数据。简而言之，浏览器会忽略任何它不理解的属性，因此尽管该属性可能尚未得到支持，即使用户使用了诸如 IE6 的浏览器，也仍然可以安全地使用这些属性。

HTML5 新增的专用文本输入类型包括：search、tel、url、email、number、color、range、日期和时间输入类型。

- **search：**搜索专用字段。与所有表单元素一样，大部分浏览器只能将搜索字段显示为一个标准的文本字段。
- **tel：**电话号码专用字段。目前，国际上没有统一的电话号码格式，因此为 tel 设置类型属性似乎不那么有用——我们必须在智能手机或平板电脑上进行测试才有效。
- **url：**统一资源定位专用字段。它的查验的严格程度可能超出我们的想象，“www.example.com”会遭受拒绝(如图 1-21 所示)，因为它不是一个有效的 URL。它要求一个以 http://等开头的完全合格的 URL。



图 1-21　Chrome 拒绝一个未加前导 http://的 URL

- **email：**email 专用字段。我们设置输入文本类型为 email，如果输入内容包含非法字符或没有包含一个@标记，浏览器将阻止表单的提交。在移动设备上，当用户单击该字段时，移动设备将显示相应的键盘。默认情况下，它能够阻止用户在该字段中填入多个 email 地址。除非我们在<input>标签中设置如下所示的 multiple 属性，否则只允许使用一个 email 地址：

```
<input type="email" name="email" id="email" multiple>
```

- **number：**数值专用字段。创建一个适用于数值输入的字段。 最新版本的主流桌面浏览器均显示一个与图 2-22 类似的能够翻动数字的输入框。单击输入框右侧的向上箭头(或者按上方向键)能够增大字段中的数字，而单击向下箭头(或者按下方向键)能够减小字段中的数字。在移动设备上，单击该字段将显示一个数字键盘，但我们也可以切换到数字字母输入方式。

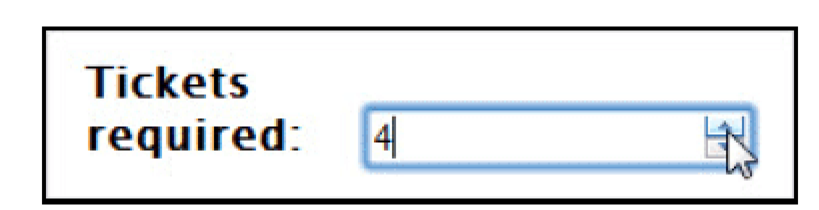


图 1-22　数字翻动输入框使得更便于输入数字值

- **color：**颜色选择器专用字段。在 Firefox、Chrome 浏览器中的显示效果如图 1-23 所示。

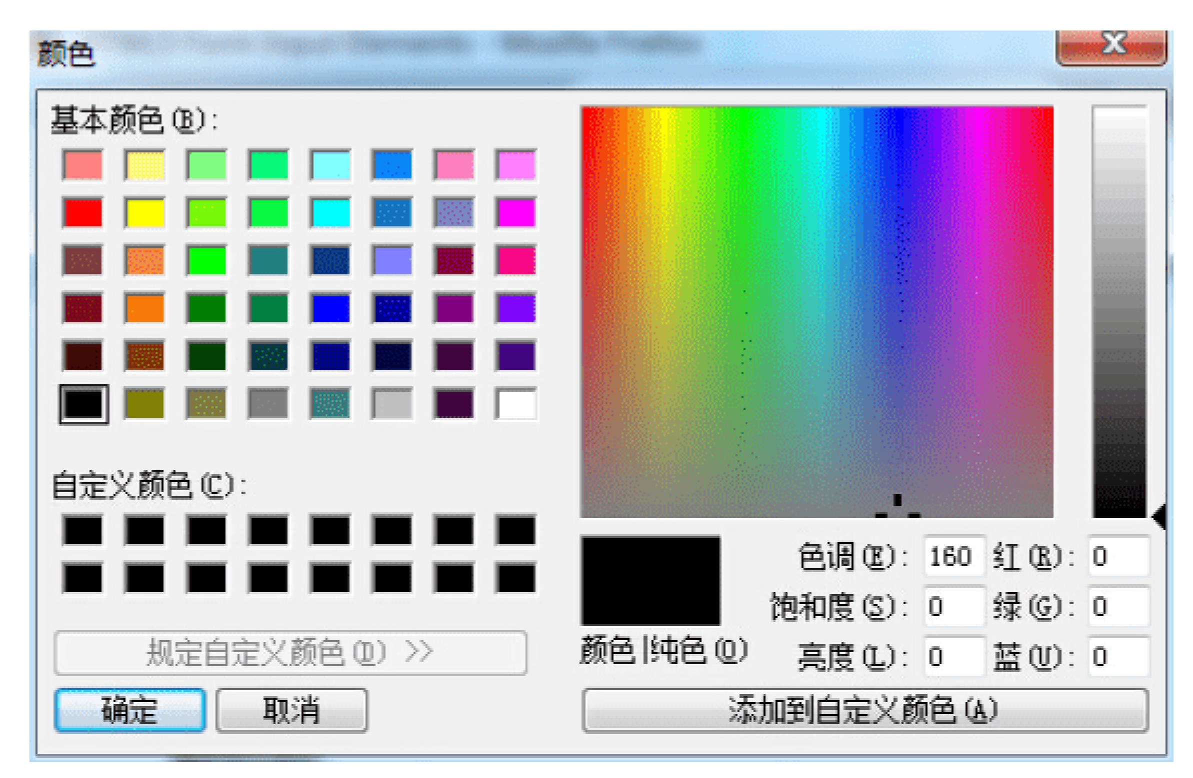


图 1-23　color 字段在 Firefox、Chrome 浏览器中的显示效果

在 Android 设备上的显示效果如图 1-24 所示。



图 1-24　color 字段在 Android 设备上的显示效果

- **range：** 数值滑动条专用字段。它能够创建一个允许用户选择数值的滑动条，显示效果如图 1-25 所示。

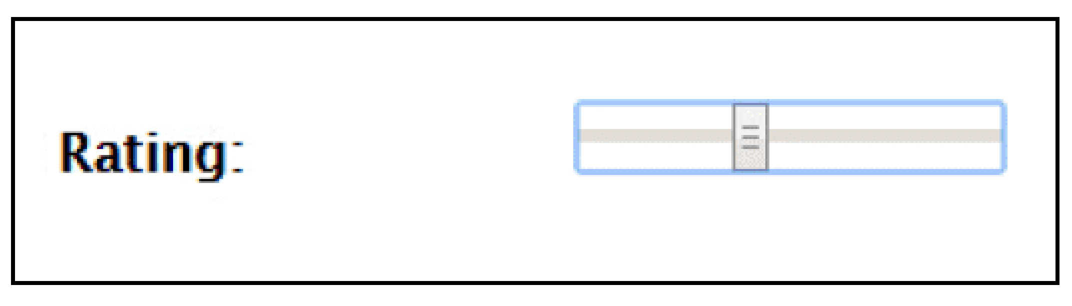


图 1-25　range 字段的显示效果

数值滑动条适合于对值的要求不是太精确的情况，因为通过滑动来得到一个精确的值会比较难操作，但我们可以进一步设置默认的范围和递增值来达到更好的效果，比如下面的例子：

```
<input type="range" name="rating" id="rating" min="0" max="25" step="5"
      value="0">
```

通过设置 mix 和 max 属性，范围被限制为 0～25，通过设置 step 使得步长为 5。

那么如何显示滑动条的值呢？这里可以使用<output>元素并在滑动条字段上使用 onchange 属性来实现，output 元素是 HTML5 中新增的、专门用于不同类型的输出的元素。完整的代

码如下：

```
<input type="range" name="rating" id="rating" min="0" max="25" step="5"
    value="0" onchange="displayRate.value=this.value">
<output name="displayRate" for="rating"></output>
```

显示效果如图 1-26 所示。

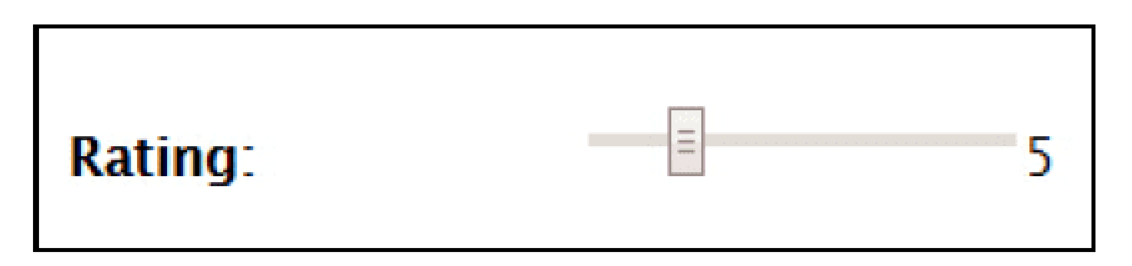


图 1-26　<output>元素的显示效果

<textarea>元素定义一个文本区域(textarea)，它是一个多行的文本输入区域。用户可以在此文本区域中写文本。在文本区域中，可以输入无限数量的文本。文本区域中的默认字体是等宽字体(fixed pitch)。它的语法为：

```
<textarea name="文本域名称" value="文本域默认值" rows="行数" cols="列数">
这里是文本域中的文本
</textarea>
```

该元素属性的含义如表 1-5 所示。

表 1-5　<textarea>元素的属性

属　性	描　述
autofocus	规定在页面加载后文本区域自动获得焦点
cols	规定文本区域内的可见列数
disabled	规定禁用该文本区域
form	规定文本区域所属的表单
maxlength	规定文本区域的最大字符数
name	规定文本区域的名称
placeholder	规定描述文本区域预期值的简短提示
readonly	规定文本区域为只读
required	规定文本区域是必填的
rows	规定文本区域内的可见行数
wrap	规定当在表单中提交时，文本区域中的文本如何换行

通常情况下，当用户在输入文本区域中键入文本后，浏览器会将它们按照键入时的状态发送给服务器。只在用户按下 Enter 键的地方生成换行。如果我们希望启动自动换行功能，当用户键入的一行文本长于文本区域的宽度时，浏览器会自动将多余的文字移动到下一行。wrap 属性的取值有 SOFT、HARD、OFF。比如，我们分别创建含 4 个不同的 wrap 属性值的<areatext>元素，输入相同的内容，将<textarea>元素的内容提交到服务器端，然后原样输出

内容，以查看其区别。

textarea.jsp 代码如下：

```
<%@ page language="java" import="java.util.*,java.text.*"
    pageEncoding="utf-8" %>
<html>
<head>
    <title>JS</title>
    <meta charset="utf-8">
</head>
<body>
<form action="textwrap.jsp" method="GET">
<table border="" cellpadding="5">
    <tbody>
        <tr>
          <td align="CENTER">this code</td>
          <td align="CENTER">produces this</td>
      </tr>
      <tr valign="TOP">
          <td bgcolor="#CCCCCC"><pre>&lt;TEXTAREA NAME="SOFT" COLS=25 ROWS=5
             WRAP=SOFT</em>&gt;</pre>
          </td>
          <td>
             <textarea name="SOFT" cols="25" rows="5" wrap="SOFT">Darkness
                 cannot drive out darkness; only light can do that. Hate cannot
                 drive out hate; only love can do that.
             </textarea>
          </td>
      </tr>
      <tr valign="TOP">
          <td bgcolor="#CCCCCC"><pre>&lt;TEXTAREA NAME="HARD" COLS=25 ROWS=5
             WRAP=HARD</em>&gt;</pre>
          </td>
          <td>
             <textarea name="HARD" cols="25" rows="5" wrap="HARD">Darkness
                 cannot drive out darkness; only light can do that. Hate cannot
                 drive out hate; only love can do that.
             </textarea>
          </td>
      </tr>
      <tr valign="TOP">
          <td bgcolor="#CCCCCC"><pre>&lt;TEXTAREA NAME="OFF" COLS=25 ROWS=5
              WRAP=OFF</em>&gt;</pre>
          </td>
          <td>
             <textarea name="OFF" cols="25" rows="5" wrap="OFF">Darkness
                 cannot drive out darkness; only light can do that. Hate cannot
                 drive out hate; only love can do that.
             </textarea>
          </td>
      </tr>
```

```
        <tr valign="TOP">
            <td>默认值
                <pre class="exampletext">&lt;TEXTAREA NAME="NONE" COLS=25
                      ROWS=5&gt;
                </pre>
            </td>
            <td>
                <textarea name="NONE" cols="25" rows="5">Darkness cannot drive
                     out darkness; only light can do that. Hate cannot drive out
                     hate; only love can do that.
                </textarea>
            </td>
        </tr>
        <tr valign="TOP">
            <td> </td>
            <td align="CENTER"><input type="SUBMIT" value="Submit"></td>
        </tr>
        </tbody>
    </table>
</form>
</BODY>
</HTML>
```

运行效果如图 1-27 所示。

this code	produces this
<TEXTAREA NAME=“SOFT” COLS=25 ROWS=5 WRAP=SOFT>	Darkness cannot drive out darkness; only light can do that. Hate cannot drive out hate; only love can do that.
<TEXTAREA NAME=“HARD” COLS=25 ROWS=5 WRAP=HARD>	Darkness cannot drive out darkness; only light can do that. Hate cannot drive out hate; only love can do that.
<TEXTAREA NAME=“OFF” COLS=25 ROWS=5 WRAP=OFF>	Darkness cannot drive out dar
默认值 <TEXTAREA NAME=“NONE” COLS=25 ROWS=5>	Darkness cannot drive out darkness; only light can do that. Hate cannot drive out hate; only love can do that.
	Submit

图 1-27　<textarea>元素的运行效果

单击 Submit 按钮后，将表单提交到 textwrap.jsp 页面，代码如下：

```
<%@ page language="java" import="java.util.*,java.text.*"
    pageEncoding="utf-8" %>
<html>
<head>
```

```
    <title>textarea</title>
</head>
<body>
<%
    out.println("<h1>SOFT</h1>");
    out.println("<pre>");
    out.println(request.getParameter("SOFT"));
    out.println("</pre>");
    out.println("<h1>HARD</h1>");
    out.println("<pre>");
    out.println(request.getParameter("HARD"));
    out.println("</pre>");
    out.println("<h1>OFF</h1>");
    out.println("<pre>");
    out.println(request.getParameter("OFF"));
    out.println("</pre>");
    out.println("<h1>NONE</h1>");
    out.println("<pre>");
    out.println(request.getParameter("NONE"));
    out.println("</pre>");
%>
</body>
</html>
```

运行结果如图 1-28 所示。

```
SOFT

Darkness cannot drive out darkness; only light can do that. Hate cannot drive out hate; only love can do that.

HARD

Darkness cannot drive out
darkness; only light can do
that. Hate cannot drive out
hate; only love can do that.

OFF

Darkness cannot drive out darkness; only light can do that. Hate cannot drive out hate; only love can do that.

NONE

Darkness cannot drive out darkness; only light can do that. Hate cannot drive out hate; only love can do that.
```

图 1-28　代码运行结果

可见，使用 SOFT 值会在显示的时候自动换行，发送到服务器端的数据不会自动换行。而使用 HARD 值会在显示的时候自动换行，发送到服务器端的数据也与用户看到的数据一样带有换行符。使用 OFF 值则不会自动换行。

- **日期和时间：**HTML5 中新增加了 date 输入类型，我们先来看一下在 Chrome 浏览器中的显示效果，如图 1-29 所示。

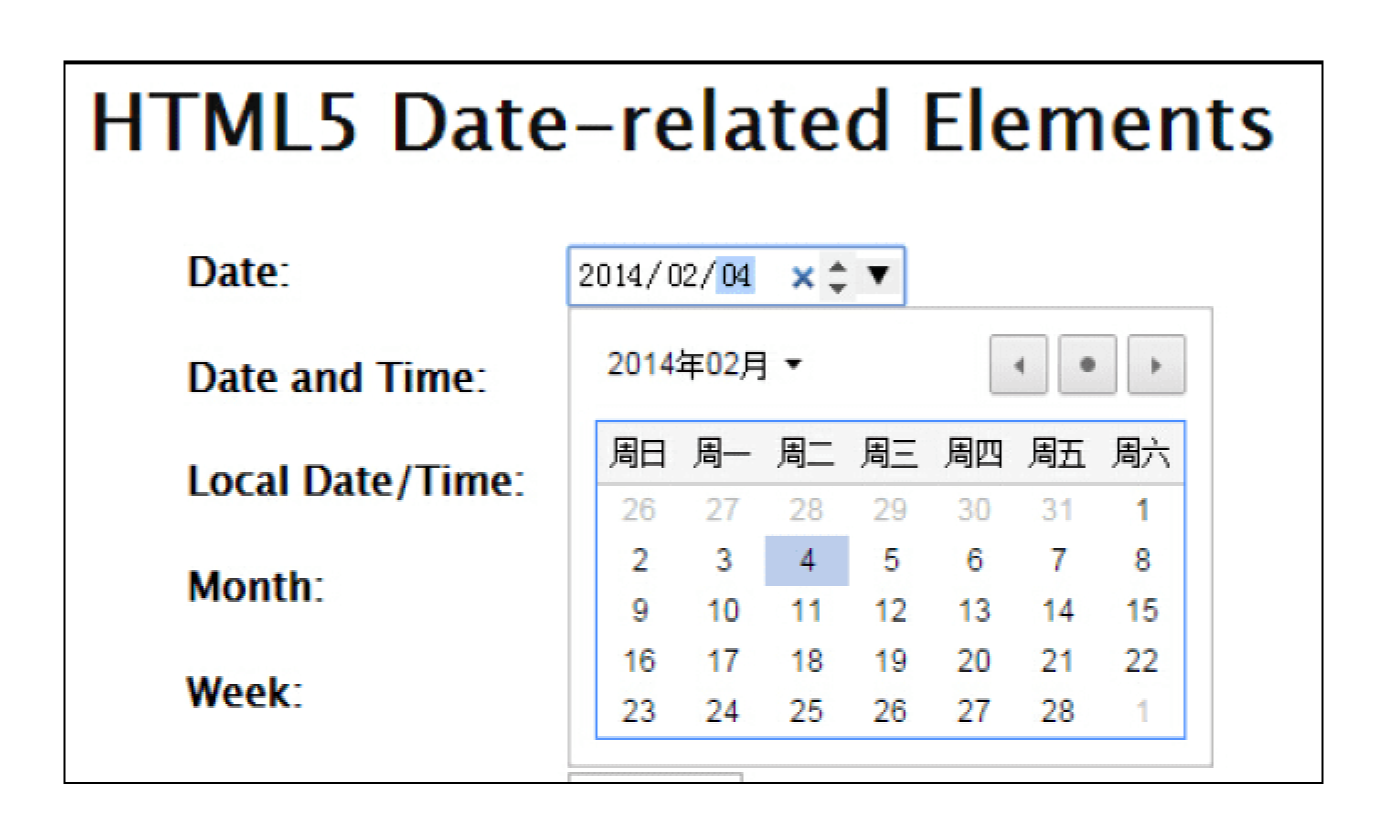


图 1-29　在 Chrome 浏览器中的显示效果

在 Android 移动设备上的显示效果如图 1-30 所示。

图 1-30　在 Android 设备上的显示效果

这是一个可翻阅的日历，默认的日期为当前日期，通过输入框中的上下箭头可以调整年月日的值。在 HTML5 之前，对于这样的页面需求，最常见的方案是用 JavaScript 日期选择组件，其实现了将日期填充到指定的输入框里的功能，这需要编写很多 JavaScript 代码。而在 HTML5 中只需要下面的代码就可以了：

```
<input type="date" name="date" id="date">
```

这样非常方便，可惜的是，现在只有 Chrome 浏览器支持这个漂亮的日历组件。如果其他浏览器也实现了 HTML5 原生的日历组件，JavaScript 版的日历组件将退出历史舞台。

此外还有其他的日期和时间输入控件，我们逐一来演示。

如果 type 类型是 month，则只能选择月份，如图 1-31 所示。

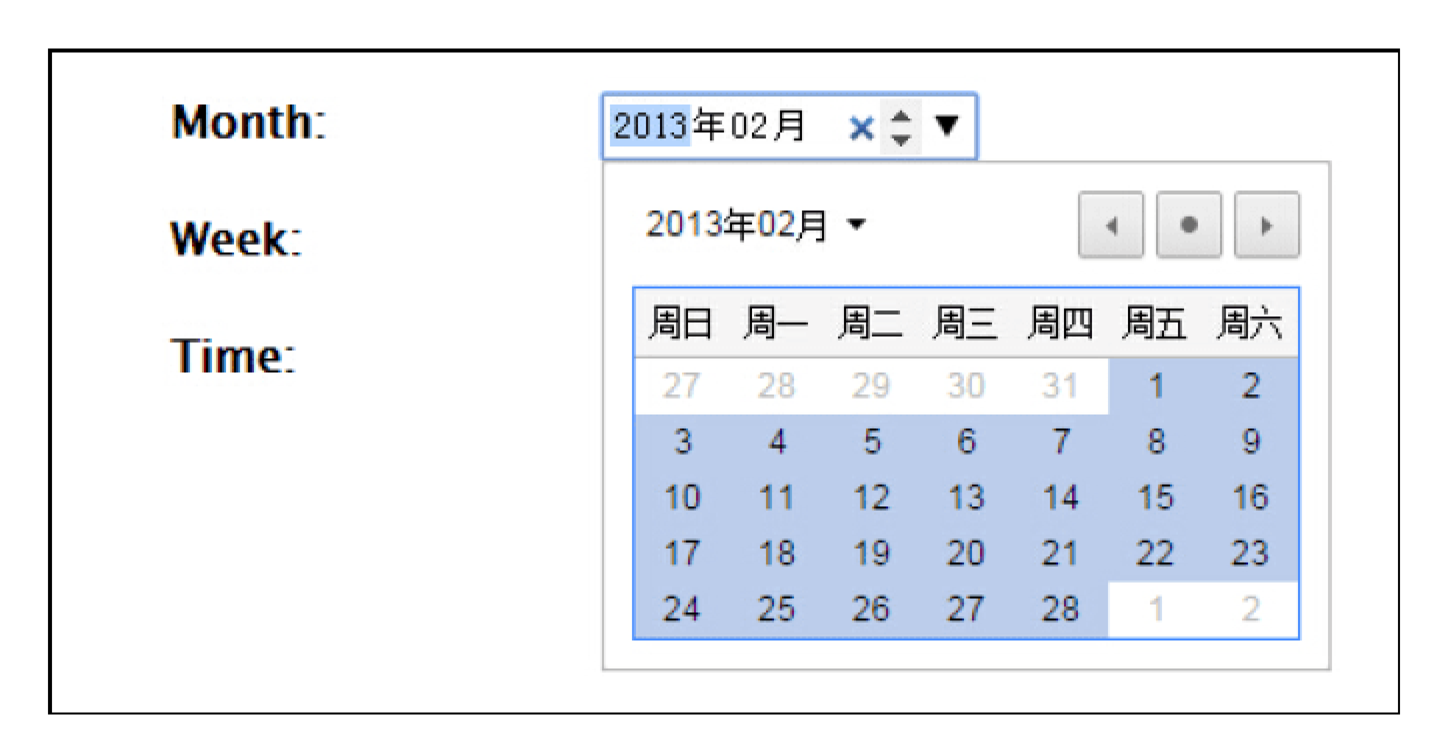


图 1-31 month 输入

如果 type 类型是 week，则只能显示周数，如图 1-32 所示。



图 1-32 week 输入

如果 type 类型是 datetime-local，则用本地时间显示日期，如图 1-33 所示。

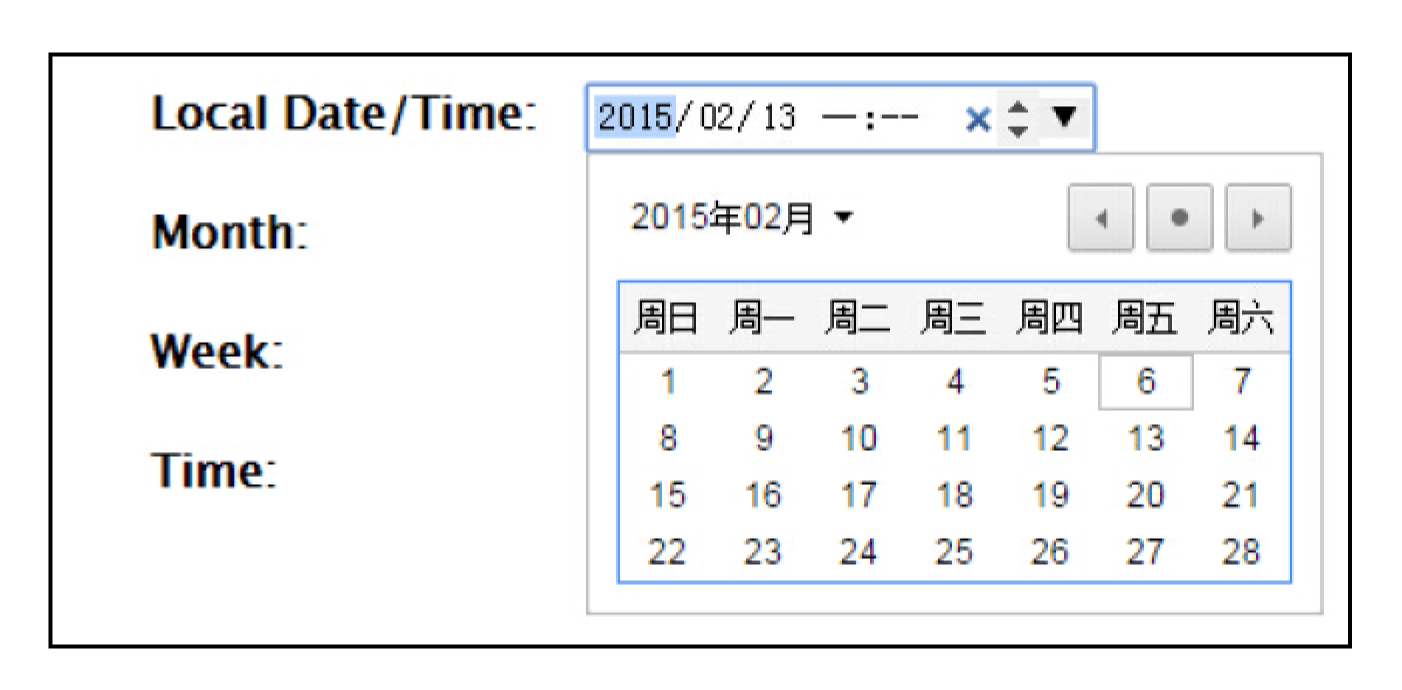


图 1-33 datetime-local 输入

如果 type 类型是 time，则只能选择时间，如图 1-34 所示。



图 1-34 time 输入

1.10　HTML5 新增的表单属性

1.10.1　autocomplete 属性

该属性规定 form 或 input 域应该拥有自动完成功能，例如浏览器能够在字段中显示一列最近输入的值(如图 1-35 所示)。默认自动完成功能是打开的。

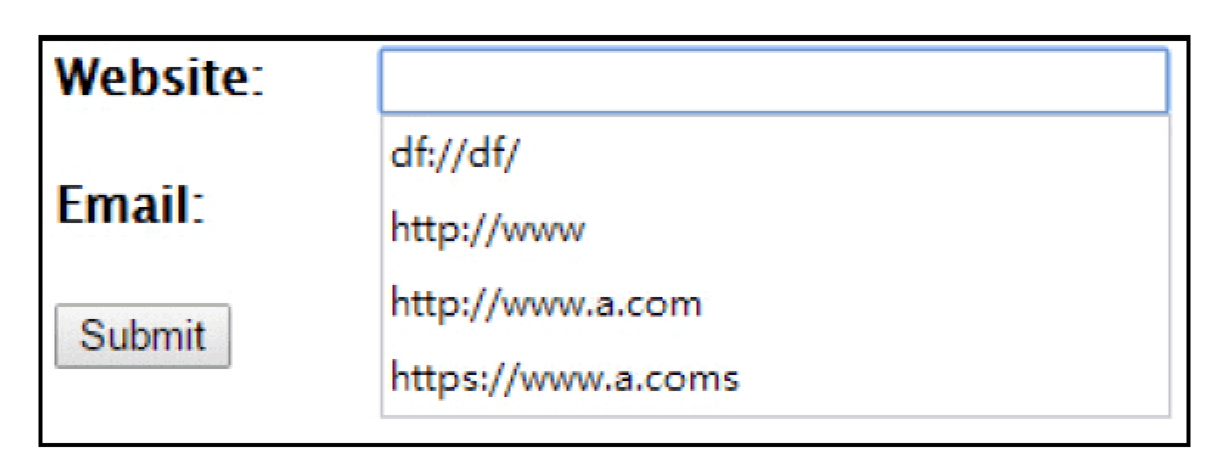


图 1-35　autocomplete 属性

这一行为因为便捷而受欢迎，但有些情况下却是危险行为，比如登录名与密码信息的预填可能是很严重的安全漏洞。为了针对整个表单关闭自动完成功能，我们可以将 autocomplete="off" 添加到打开的 <form> 标签中。为了针对个别元素关闭自动完成功能，将 autocomplete="off" 添加到相应的元素中。添加到元素的 autocomplete 属性会覆盖 form 表单的属性。

1.10.2　required 属性

该属性规定必须在提交之前填写输入域(不能为空)。经常会有表单中的信息是要求必须填写的。例如登录名，可以使用 required 属性确保必填表单中有内容，表单才可以提交。代码如下：

```
<input id="login_name" name="login_name" type="text" required/>
```

1.10.3　multiple 属性

该属性规定输入域中可选择多个值。它适用于以下类型的 <input> 元素：email 和 file。默认未打开该功能。比如：

```
<input type="file" name="img" multiple="multiple" />
```

1.10.4　autofocus 属性

该属性规定在页面加载时，拥有该属性的输入元素自动获得焦点。它适用于所有 <input> 元素的类型。我们应该在一个表单中使用 autofocus 一次。例如：

```
<input type="text" name="user_name"  autofocus="autofocus" />
```

1.10.5　formnovalidate 属性

该属性覆盖表单的 novalidate 属性。它适用于以下类型的 <input> 元素：submit 和 image。比如：

```
<form action="submit.html" method="post" name="form1">
        <p>
        <!-- 保存草稿 -->
        <input type="submit" name="draft" id="draft" value="Save Draft"
            formnovalidate formaction="draft.html">
        <!-- 提交数据 -->
        <input type="submit" name="send" id="send" value="Submit"
            formmethod="get">
    </p>
</form>
```

上面的例子有一个保存草稿的按钮，我们不希望保存草稿时对表单进行验证。还有些情况我们希望用自定义的验证方法验证表单，也可以在按钮控件上设置 formnovalidate 属性。

此外，类似的 formaction 、formenctype 、formmethod、formtarget 属性也能够覆盖表单属性。

1.10.6　placeholder 属性

该属性会在输入框中显示所期望的值或提示信息，提示信息会在输入框获得焦点后消失。在以前，会使用 value 属性提示用户输入示例。这种技术的缺点是，用户未必总会清空示例输入内容，因此该文本经常会作为数据的一部分被包含进来发送到服务器端。使用 placeholder 属性则可解决这个问题。例如：

```
<body>
<h1>Placeholder</h1>
<form method="post" name="form1">
    <p>
        <label for="zip">Zip Code:</label>
        <input type="text" name="zip" id="zip" placeholder="5 digits">
    </p>
    <p>
        <label for="arrival">Arrival Date:</label>
        <input type="text" name="arrival" id="arrival"
            placeholder="MM/DD/YYYY">
    </p>
    <p>
        <input type="submit" name="send" id="send" value="Submit">
    </p>
</form>
</body>
```

运行效果如图 1-36 所示。

图 1-36　placeholder 属性

1.10.7　list 属性

该属性规定输入元素的选项列表。我们在第 1 章学习了<datalist>元素，这里通过输入元素的 list 属性绑定 datalist 来定义可能的值。比如：

```
Webpage: <input type="url" list="urlList" name="link" />
<datalist id="urlList">
<option label="baidu" value="http://www.baidu.com" />
<option label="Google" value="http://www.google.com" />
<option label="Microsoft" value="http://www.microsoft.com" />
</datalist>
```

运行效果如图 1-37 所示。

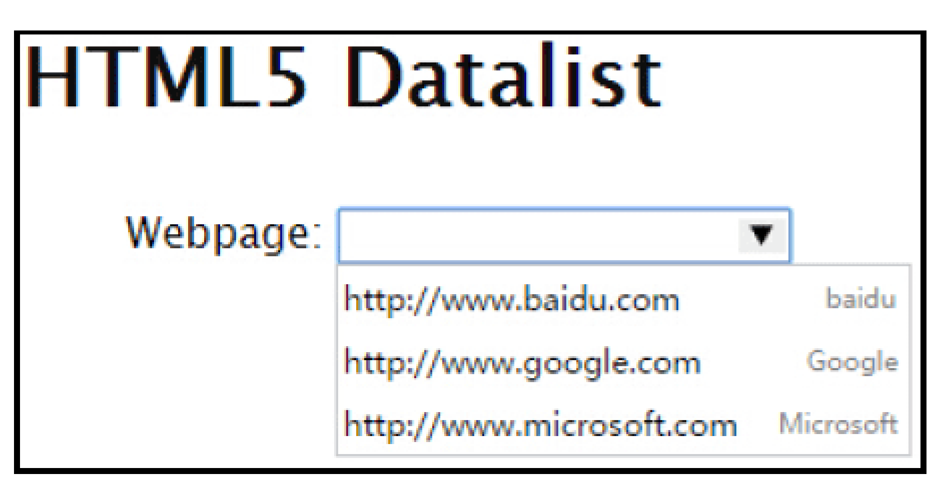


图 1-37　list 属性

1.10.8　pattern 属性

该属性规定用于验证输入元素的模式。也就是我们可以用自定义的方式验证输入内容。pattern 属性必须是一个正则表达式，根据要求建立正则表达式不是一件容易的事，我们可以通过搜索引擎或者 http://regexlib.com/的知识库，查看前人书写过的正则表达式。比如：

```
<input id="username" name="username" type="text" placeholder="4-10 个英文字母"
    pattern="[A-Za-z]{4,10}" required="required" autofocus />
```

1. 案例概述

在本案例中，我们将修改人人微博网站的用户个人信息页面。该页面的功能是显示用户个人信息和修改用户个人信息。原 HTML4 版本的效果如图 1-20 所示。

该页面主要是在用户自己查看或修改信息时使用，“登录名”字段是在用户注册时确定的，是从服务器端读取的数据，所以不能被修改。我们要实现必输字段的验证、字段有效长

度的验证，最后通过“保存”按钮提交表单。并且新的需求要求增加“年龄”和“备用邮箱”字段，还要增加个人头像上传功能。

这里要求能够记录表单的已输入条目，以便下次输入时更快捷。

2. 显示效果

首先来看一下改版后的页面在浏览器中的显示效果，如图 1-38 所示。

图 1-38　改版后的个人信息页面

该页面需要具有自动验证的功能，当用户在某个文本框中输入无效数据后单击“保存”按钮时，页面上自动显示错误提示信息，阻止表单向服务器端提交。如何提交数据到服务器端将在后面的章节介绍。

用户选择的图像文件的尺寸可能和网站要求的缩略图的尺寸不一样，个人头像功能需要根据用户选择的图像文件生成缩略图，生成缩略图的具体细节以后再实现，只要知道将图像文件上传到服务器后，在服务器端会生成缩略图，并将缩略图的 URL 地址返回给客户端就可以了，这里先把显示用户上传图像文件和缩略图的位置预留好就可以了。用户可以单击“上传”按钮上传图像文件，然后只有在所有字段填写正确后，单击“保存”按钮才会成功地提交数据。

3. 案例分析

我们知道 form 元素的 enctype 属性表示在发送到服务器之前如何对表单数据进行编码。在包含文件上传控件的表单中，该属性必须设置为 multipart/form-data，表示以二进制的形式上传。

在这里可以使用一个 form 表单提交数据，后台服务器同时生成缩略图和保存表单数据，但这样的话，后台的逻辑代码会比较复杂；也可以使用两个表单提交数据，一个用来生成缩略图，另一个用来保存表单数据。我们采用第二种方法，使用<input>元素的 form 属性指定其属于哪一个表单。

该页面的所有控件的信息如表 1-6 所示。

表 1-6　个人信息页面控件信息

参 数 名	类　型	最大长度或最大值
登录名	text	
昵称	text	8
缩略图	img	
真实姓名	text	8
所在地	select	
性别	radio	
年龄	range	100
备用邮箱	email	
博客地址	text	
QQ	number	
MSN	text	
生日	date	

4. 代码剖析

接下来我们看一下本案例的 HTML5 代码，下面给出了主要的代码结构。

```
<div id="rightpart">
    <div id="topdown">以下信息将显示在<a href="#">个人资料页</a>，方便大家了解你
    </div>
        <!-缩略图显示区域->
          <div id="div_preview" >
                 <img id="crop_preview" src=""/>
          </div>
    <!-登录名->
    <input id="login_name" name="login_name" type="text" value=""
    maxlength="8" readonly  form="infoForm"/>
    <!-昵称->
    <input id="nick_name" name="nick_name"  type="text" value=""
    maxlength="8" required autofocus form="infoForm" maxlength="8"/>
    <!-头像上传->
     <div id="midleft2">头像：</div>
      <div id="uploader">
    <!----------------------提交图片 uploadForm -------------------------->
                <form name="" id="uploadForm" method="post"
                    enctype="multipart/form-data"  >
                <div>
```

```
                <input name="photo" id="file"  size="27" type="file"
                    accept="image/gif, image/jpeg,image/x-png"
                    form="uploadForm"/>
                <input id = "upload_but" name="upload_but" value=
                    "上传" type="submit" form="uploadForm"/>
            </div>
            </br>
            <div id="div_big_view" >
                <img id="big_view" src="" form="infoForm"/>
            </div>
        </form>
  </div>
  <!-真实姓名->
  <input id="real_name" name="real_name" type="text" value=""
      maxlength="8"
required form="infoForm"/>
<!-省市->
 <select name="province" id="province" form="infoForm"></select>
 <select name="city" id="city" form="infoForm"> </select>
<!-性别->
 <div id="midright4">
        <input name="gender"  type="radio" value="1" checked="checked"
            form="infoForm"/>男
        <input name="gender"  type="radio" value="0" form="infoForm"/>女
  </div>
<!-年龄->
  <input type="range" name="age" id="age" min="0" max="100" step="1"
      value="0" onchange="display_age.value=this.value"
      form="infoForm">
      <output name="display_age" id="display_age" for="age"
            form="infoForm"></output>
<!-邮箱->
 <input type="email" name="email" id="email" placeholder="请输入邮箱地址"
     form="infoForm">
<!-生日->
 <input type="date" name="birthday" form="infoForm"/>
<!-博客->
 <input name="blog" type="text" value="" form="infoForm"/>
<!-QQ->
 <input name="qq" type="number" value="" form="infoForm"/>
<!-MSN->
 <input name="msn" type="text" value="" form="infoForm"/>
 <!---------------------------提交数据 infoForm ------------------------->
 <form method="post" enctype="multipart/form-data" name="infoForm"
     id="infoForm">
        <button type="submit" name="send" id="send" value=""><span>保存
            </span></button>
  </form>
  </div>
```

首先增加了上传图像文件的 form 区域和缩略图预览区域，又为每个<input>元素指定了

正确类型，设置昵称为必输字段和自动得到焦点，最后增加了提交数据的 form 区域。在 HTML5 中，我们不必把所有输入元素放置于打开和关闭的<form>元素中。我们可以将表单放置于任何我们愿意放置的位置，设置输入元素的 form 属性为表单的 ID，这样输入元素就可以和一个表单关联了。比如：

```
<input name="qq" type="number" value="" form="infoForm"/>
```

本章小结

本章讨论了 HTML5 规范的历史和 HTML5 的重要特性，介绍了规范到底包括什么，不包括什么，比如新的 doctype 和字符集声明，许多新的语义化标记元素。学习了 Web 页面的内容是使用描述文档结构的元素进行标记的，包括 HTML5 新增的页眉、页脚、内容区块等主体结构元素和表示逻辑结构或附加信息的非主体结构元素。

HTML5 规定了<audio>和<video>元素包含音频和视频的标准方法，还演示了如何使用它们构建引人注目的 Web 应用。<audio>元素和<video>元素的引入，让 HTML5 应用在对媒体的使用上多了一种选择：不用插件即可播放音频和视频。此外，<audio>元素和<video>元素还提供了通用的、集成化的、可用脚本控制的 API。

Google 提出了一项计划来推进 HTML5 默认取代 Flash。Google 计划在 2016 年第 4 季度，只允许 10 个域名默认支持 Flash，其他域名默认显示 HTML5。如果依然要使用 Flash，浏览器将询问用户是否允许运行 Flash 插件。

由于有的元素只有少部分浏览器支持或者使用率极低等原因，HTML5 废除了一些元素，有的元素给出了可替代的元素。表 1-7 给出了详细内容。

表 1-7　废除的元素

废除的元素	替 代 方 案
frameset、frame、noframes	使用 iframe 或 Ajax 技术
center、font、s、strike、tt、u、basefont、big、	使用 CSS 替代的元素
rb	使用 ruby 元素替代
dir	使用 ul 元素替代

在使用 HTML5 新的输入元素创建了一个表单之后，还应该确保每个元素都设置了恰当的类型，以便让用户了解应该填写何种信息或者应该做何种选择。一方面给用户带来了一些新的体验，另一方面减少了程序开发的时间。

HTML5 Web Forms 是对目前 Web 表单的全面提升，它保持了简便易用的特性。通过书中的表单示例，我们可以很快掌握表单的新特性，当然有些新的表单属性还有待挖掘。在熟

练掌握新属性后，我们可以关注如何提升用户操作体验。

2016 年 4 月 20 日，微信发布消息称安卓版客户端 WebView 即日起 100%放量，全面升级至 X5 Blink 内核。X5 Blink 内核对 HTML5 和 CSS3 有着更好的支持、更强大的渲染能力，目前腾讯的应用，诸如微信、手机 QQ 等均已通过进行 X5 Blink 内核过渡。X5 Blink 内核将会为开发者提供更好的网页支持，对诸如动画、音视频、Touch 事件、滚动等的支持性会更好。对于微信用户来说，以后将会看到更多拥有各式各样功能的网页或小游戏等，用户的浏览体验也会更进一步。配合 TBS 2.0(腾讯浏览服务)的发布，新版微信开发者工具也随之发布，完善开发者远程调试能力，为广大前端开发者开发 Web App 提供更多可能和便利。

本章练习

1. 如何区分 HTML 和 HTML5？
2. HTML5 的页面结构与 HTML4 或更早的 HTML 相比有什么区别？
3. HTML5 中的 datalist 是什么？
4. 如何在 HTML5 页面中嵌入音频和视频?
5. HTML5 表单新增加的元素有哪些？
6. HTML5 表单如何兼容低版本浏览器？
7. 如何在 email 文本框中输入多个 email 地址？

第2章　CSS3和BootStrap

我们看到许多网站都非常漂亮，这是因为这些网站使用了 CSS3 的新特性以及类似 BootStrap 的响应式 UI 框架。

本章内容：

- 掌握 CSS3 特效
- 掌握 BootStrap 基础知识

2.1　CSS3 简介

CSS3 是对 CSS2.1 的扩展，并且完全向后兼容，它带来了让开发人员期望已久的新特性，包括圆角边框、边框阴影、过渡效果和动画效果。

从 2002 年到 2011 年，CSS2 的发布用了 9 年时间，而 CSS3 为了在规范化的进程中走得更快，采取了一种模块化的方式进行规范的设计，不同的模块增强了之前版本规范中所引入的功能。每个模块在 W3C 中会进行分级，比如“提案建议”、“候选建议”、“建议”等，只有符合官方的一定标准，才被称为成熟的 CSS3 模块。

本章会以小案例的形式介绍 CSS3 的相关技术点。

2.1.1　CSS3 边框

假设要实现图 2-1 所示的边框效果。



图 2-1　边框效果

我们看到矩形框的 4 个角是圆角的，还有边框阴影。在 CSS3 之前，实现此效果难度较大，十分复杂，但在有了 CSS3 之后，实现起来就很简单。

border-radius 属性就能实现圆角边框，box-shadow 属性就能实现边框阴影。具体实现代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title> CSS3 Corners 案例</title>
<style type="text/css">
    .box {
        width: 300px;
        height: 150px;
        background: #ffb6c1;
        border: 2px solid #f08080;
        border-radius: 20px;
        box-shadow: 15px 15px 5px #888245;
    }
</style>
</head>
<body>
   <div class="box"></div>
</body>
</html>
```

2.1.2　CSS3 多列

在 CSS3 中如何实现多列布局呢？比如报纸布局(参见图 2-2)。

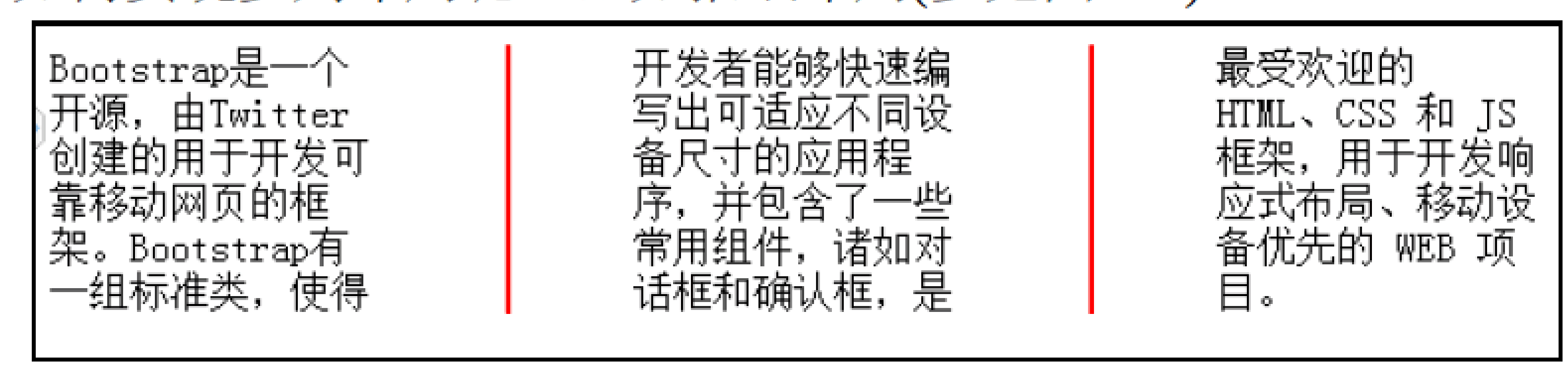


图 2-2　报纸布局

在完成图 2-2 所示布局前，需要掌握一些预备知识。CSS3 中关于多列的属性主要有如下三个：

- column-count：指定元素的列数
- column-rule：指定列之间的差距
- column-gap：设置列之间的宽度、样式和颜色

读者可以尝试完成这个案例，在这里给出实现代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Example of Setting CSS3 Column Rules</title>
<style type="text/css">
    p {
        /* Chrome, Safari, Opera */
        -webkit-column-count: 3;
        -webkit-column-gap: 100px;
        -webkit-column-rule: 2px solid red;
```

```
            /* Firefox */
            -moz-column-count: 3;
            -moz-column-gap: 100px;
            -moz-column-rule: 2px solid red;
            /* Standard syntax */
            column-count: 3;
            column-gap: 100px;
            column-rule: 2px solid red;
        }
    </style>
    </head>
    <body>
        <p>BootStrap 是一个开源，由 Twitter 创建的用于开发可靠移动网页的框架。BootStrap 有一组标准类，使得开发者能够快速编写出可适应不同设备尺寸的应用程序，并包含了一些常用组件，诸如对话框和确认框，是最受欢迎的 HTML、CSS 和 JS 框架，用于开发响应式布局、移动设备优先的 WEB 项目。</p>
    </body>
    </html>
```

2.1.3 CSS3 Box-Sizing

默认情况下，元素渲染时的尺寸取决于宽度、高度、边框和边距。比如，你对一个 100% 宽度的 div 元素使用 padding 和 border 属性的话，水平的滚动条就会出现，例如：

```
.box {
    width: 100%;
    padding: 20px;
    border: 5px solid #f08080;
}
```

这是前端开发者经常遇见的问题，CSS3 中使用 box-sizing 解决了这个问题，使得布局更加简单和自然。box-sizing 属性值如表 2-1 所示。

表 2-1 box-sizing 属性值

值	描 述
content-box	这是由 CSS2.1 规定的宽度及高度行为 宽度和高度分别应用到元素的内容框 在宽度和高度之外绘制元素的内边距和边框
border-box	为元素设定的宽度和高度决定了元素的边框盒。也就是说，为元素指定的任何内边距和边框都将在已设定的宽度和高度内进行绘制。通过从已设定的宽度和高度分别减去边框和内边距才能得到内容的宽度和高度
inherit	规定应从父元素继承 box-sizing 属性值

2.1.4 CSS3 背景

CSS3 中的 background-image 属性允许我们为背景设置多张图片，应该确保图片有一定

的透明度才能看出多张图片的效果。例如，假设有图 2-3 和图 2-4 所示的两张背景图。

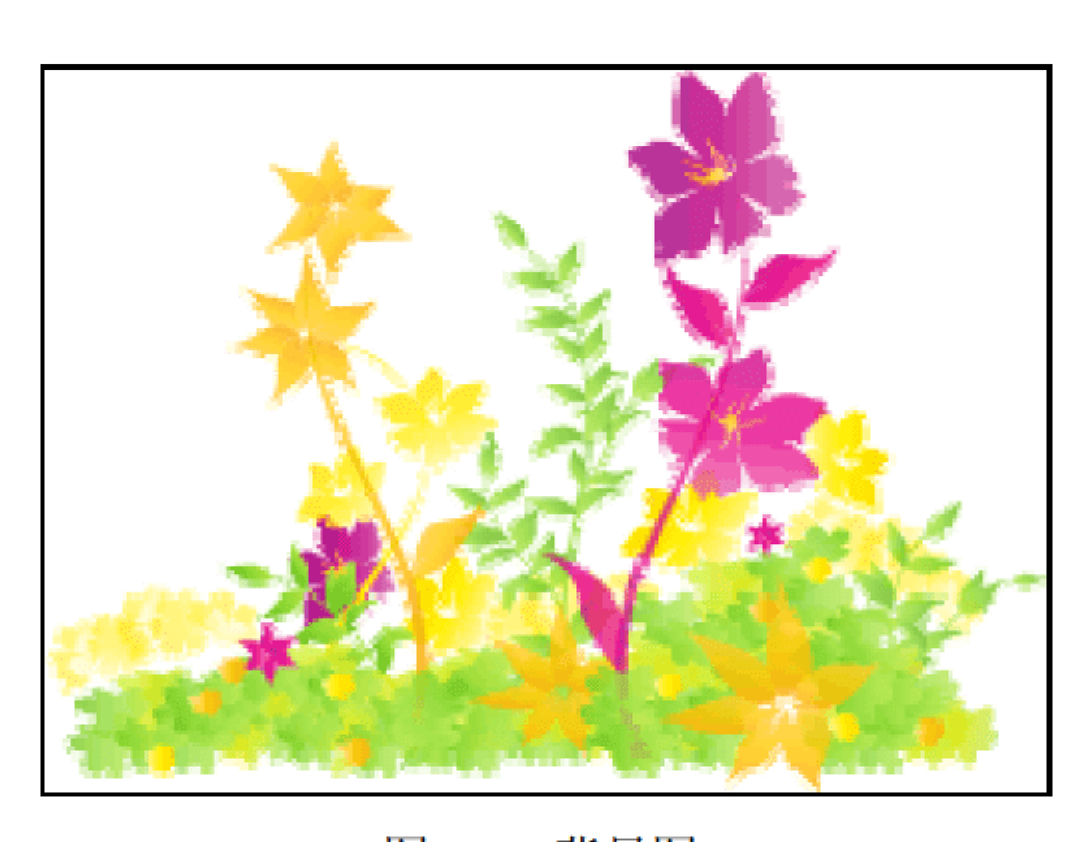


图 2-3　背景图 1

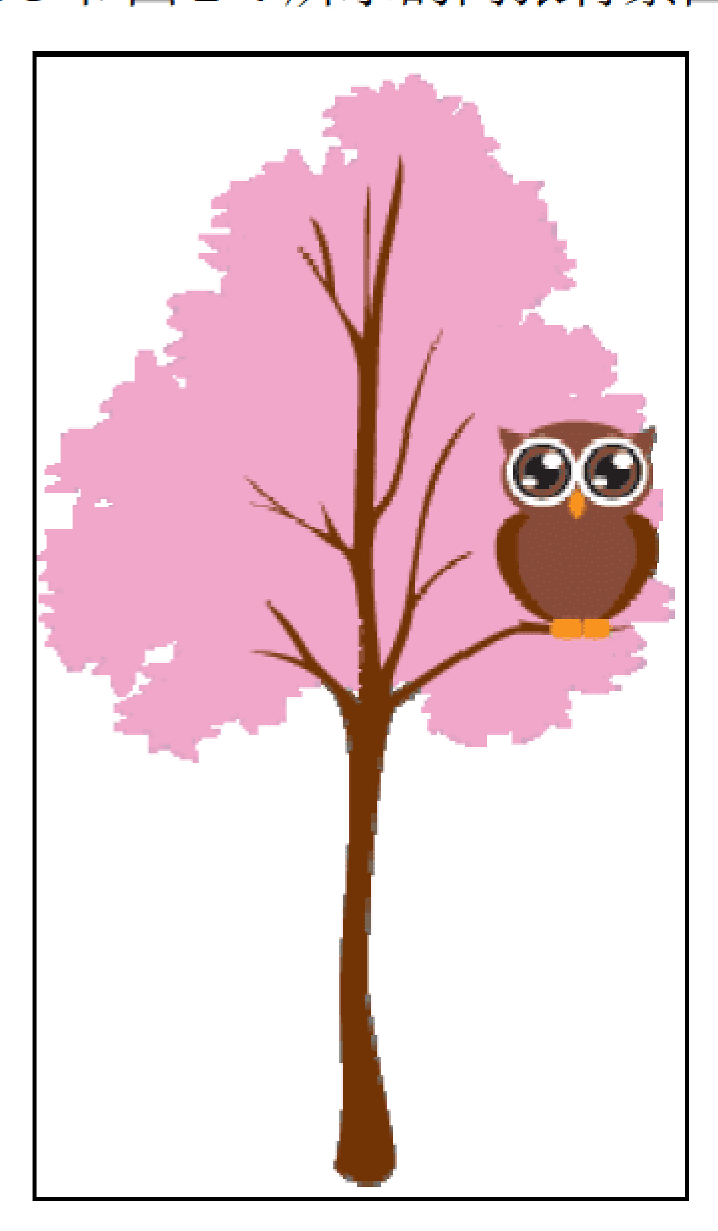


图 2-4　背景图 2

在网页中使用如下样式代码：

```
body {
    background-image:url("img_tree.gif"),url("img_flwr.gif");
    background-color: "#cccccc";
 }
```

运行效果如图 2-5 所示。

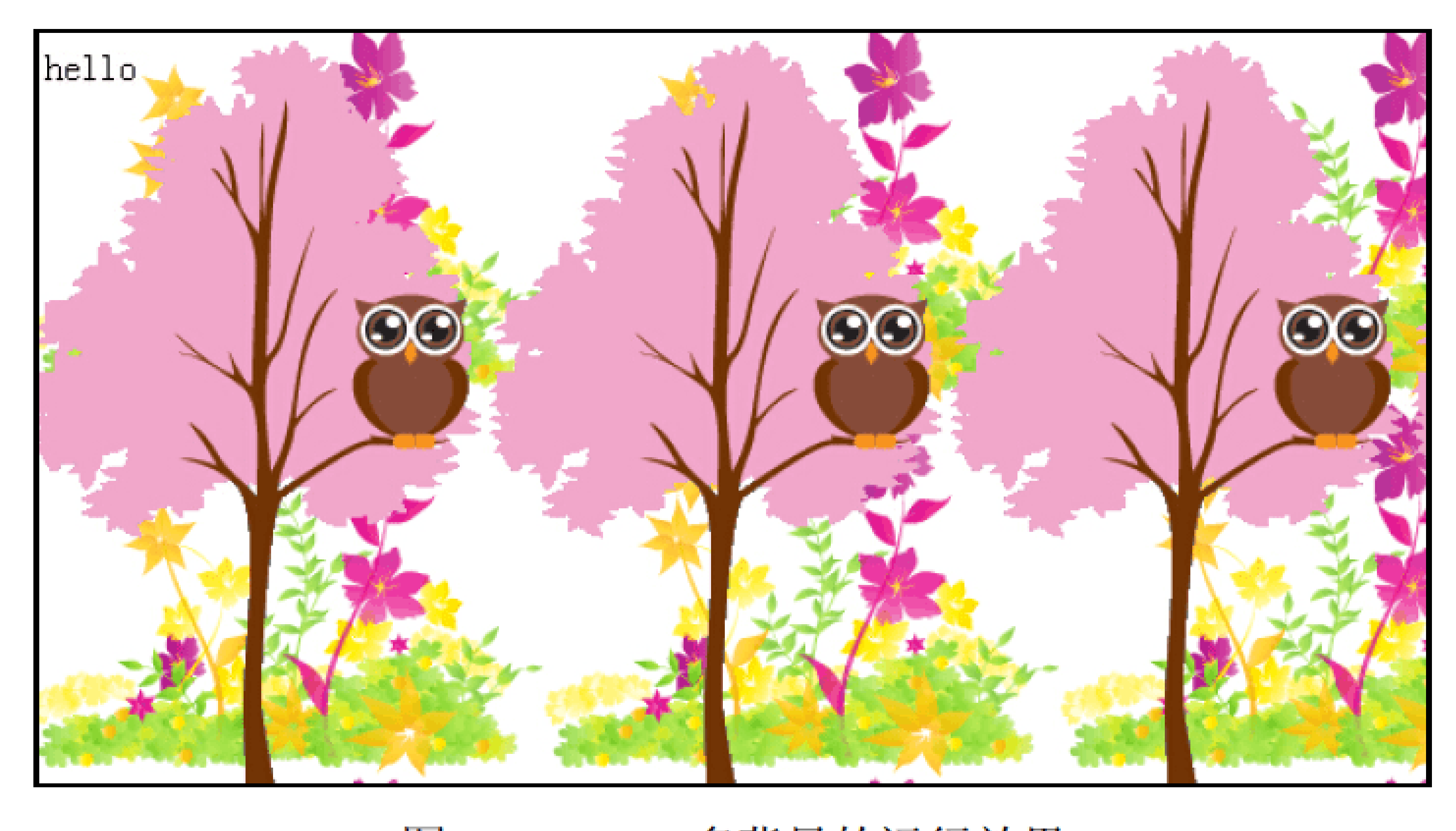


图 2-5　CSS3 多背景的运行效果

对于背景图片的设置还有另一个有用的属性——background-size 属性规定背景图片的尺寸，我们可以使用数字、百分比、内置字符串选项来设置 background-size 的值。具体如表 2-2 所示。

表 2-2　background-size 属性值

值	描　述
数字值	设置背景图像的高度和宽度。第一个值设置宽度，第二个值设置高度。如果只设置一个值，第二个值会被设置为 auto
percentage	以父元素的百分比来设置背景图像的宽度和高度。第一个值设置宽度，第二个值设置高度。如果只设置一个值，第二个值会被设置为 auto
cover	把背景图像扩展至足够大，以使背景图像完全覆盖背景区域。背景图像的某些部分也许无法显示在背景定位区域
contain	把背景图像扩展至最大尺寸，以使其宽度和高度完全适应内容区域

例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>CSS3</title>
<style type="text/css">
 .block {
   width:450px;
   height: 450px;
   background-color: "#cccccc";
  }
  #div1 {
   background-image: url("../_assets/img/phone_onex.png");
   background-size: cover;
   background-repeat:no-repeat;
   float: left;
  }
  #div2 {
   background-image: url("../_assets/img/phone_onex.png");
   background-size: contain;
   background-repeat:no-repeat;
   float: left;
  }
</style>
</head>
<body>
<div id="div1" class="block"></div>
<div id="div2" class="block"></div>
<script>
</script>
</body>
</html>
```

运行效果如图 2-6 所示。



图 2-6　background-size 的运行效果

2.1.5　CSS3 2D 转换

CSS3 的 2D 转换功能能够改变元素的形状、尺寸和位置，因此我们能够使用 CSS3 制作出对元素进行移动、缩放、转动、拉伸的效果，再结合 jQuery 的动画特效就能做出意想不到的效果。2D 变形模块引入了新的 transform 属性，它允许对盒子进行旋转、平移以及缩放而不会改变其在文档流中的位置。下面我们来一一学习这个属性的常用方法。

通过 translate 方法将元素上/下、左/右移动，基于 translate 方法的还有 translateX 和 translateY 方法，例如：

```
<button id = "trans">移动</button>
.translate {
  color: #009999;
  border: #009999;
  -webkit-transform: translate(200px, 200px);
  -moz-transform: translate(200px, 200px);
  -ms-transform:  translate(200px, 200px);
  -o-transform: translate(200px, 200px);
  transform: translate(200px, 200px);
}
<script>
$('#trans').click(function(event) {
  $(this).addClass('translate');
});
</script>
```

由于有的浏览器并不支持所有的变换效果，因此我们使用了带浏览器前缀的 transform 属性以兼容各个浏览器，IE9、Firefox 和 Opera 分别对应-ms-transform、-moz-transform 和-o-transform，Safari 和 Chrome 对应-webkit-transform，一般使用-webkit-transform 和-moz-transform 就可以了。注意我们把不带前缀的 transform 放在最后，这是因为如果有浏览器支持 transform 属性，就能够优先使用不带前缀的 transform 属性进行处理。

上述代码的运行效果如图 2-7 所示。

rotate 方法可以围绕 X 轴或 Y 轴进行旋转，并指定角度值。例如：

```
<button class="rotate">旋转</button>
.rotate:hover{
  -webkit-transform: rotate(90deg);
  -moz-transform: rotate(90deg);
  transform: rotate(90deg);
}
```

运行效果如图 2-8 所示。

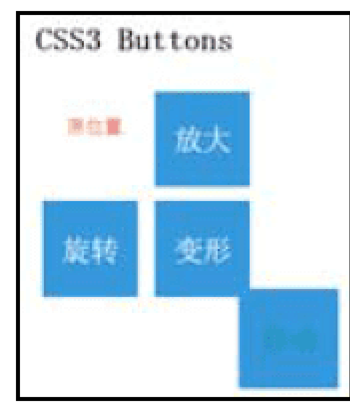


图 2-7 移动效果

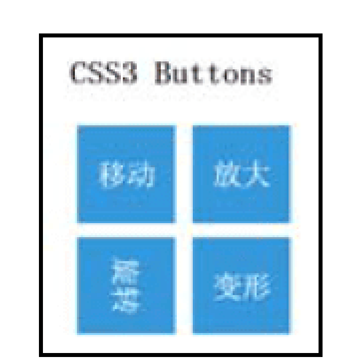


图 2-8　旋转效果

scale 方法对元素进行缩放，该属性以 1 的倍数进行指定，scale(2)告知浏览器将元素放大至两倍大小。基于此方法的还有 scaleX 和 scaleY 方法。例如：

```
<button class="boom">放大</button>
.boom:hover{
  color: #009999;
  border: #009999;
  -webkit-transform: scale(2, 2);
  -moz-transform: scale(2, 2);
  transform: scale(2, 2);
}
```

运行效果如图 2-9 所示。

skew 方法将元素翻转给定的角度，skew(30deg,10deg)围绕 X 轴把元素翻转 30 度，围绕 Y 轴翻转 10 度。例如：

```
 <button class="deform">变形</button>
.deform:hover{
  -webkit-transform: skew(15deg,15deg);
  -moz-transform: skew(15deg,15deg);
  transform: skew(15deg,15deg);
}
```

运行效果如图 2-10 所示。



图 2-9　缩放效果

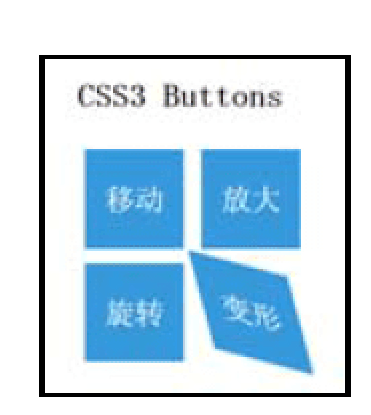


图 2-10　变形效果

2.1.6　CSS3 过渡

通过 CSS3 过渡，我们可以在不使用 Flash 动画或 JavaScript 的情况下，在元素从一种样式变换为另一种样式时为元素添加效果。在引入 CSS 动画之前，由于实现动画的 JavaScript 指令不尽相同，浏览器无法优化其正在执行的 JavaScript 指令，以得到对人眼更平滑的动画效果。

想为元素赋予动画，需要指定动画、时间以及动画过程中变动的单位等属性。CSS3 必须指定两项内容：指定希望把效果添加到哪个 CSS 属性和指定效果的时长。当我们改变指定的 CSS 属性时，就会得到动画效果。例如：

```
div
{
transition: width 3s;
-moz-transition: width 3s;   /* Firefox 4 */
-webkit-transition: width 3s;    /* Safari 和 Chrome */
-o-transition: width 3s; /* Opera */
}
```

上面的代码给出了应用于 width 属性的过渡效果，时长为 3 秒。如果要添加多个属性，需要由逗号隔开。例如：

```
transition: width 3s, height 3s;
```

也可以对所有属性赋予动画效果，例如：

```
transition: all 3s;
```

我们还可以单独指定 CSS3 的过渡属性，如表 2-3 所示。

表 2-3　过渡属性

属　　性	描　　述
transition-property	应用过渡的 CSS 属性的名称
transition-duration	过渡效果的时间，默认是 0
transition-timing-function	过渡效果的时间曲线，默认是 ease，包括 ease、ease-in、ease-out、ease-in-out、cubic-bezier
transition-delay	过渡效果的延迟执行时间，默认是 0

比如：

```
div
{
transition-property: width;
transition-duration: 3s;
transition-timing-function: linear;
transition-delay: 2s;
}
```

对于上一节中 2D 变换的例子，我们可以添加过渡效果，并可以组合使用多个 2D 变换以得到想要的效果。例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>CSS3</title>
<style type="text/css">
 #container{
  width: 330px;
  margin: 25px auto;
}
button{
  width: 100px;
  height: 100px;
  color: #ffffff;
  background-color: #3498db;
  font-family:Open Sans Condensed;
  text-decoration: none;
  font-size: 1.8em;
  border: 2px #ffffff solid;
  margin: 10px 0 0 5px;
  //设置过渡效果
  -webkit-transition: all 0.5s;
  -moz-transition: all 0.5s;
  transition: all 0.5s;
}
/*魔力从这里开始*/
button:hover{
  cursor: pointer;
}
.highlight:hover{
  background-color: #FFffff;
  color: #3498db;
}
.fade:hover{
  border: 0px;
```

```
    color: #009999;
    opacity:0;
  }
  .boom:hover{
    color: #009999;
    border: #009999;
    opacity: 0;
    -webkit-transform: scale(2, 2);
    -moz-transform: scale(2, 2);
    transform: scale(2, 2);
  }
  .focus:hover{
    border-width: 20px;
  }
  .rotate:hover{
    -webkit-transform: rotate(90deg);
    -moz-transform: rotate(90deg);
    transform: rotate(90deg);
  }
  .knit:hover{
    height: 0px;
  }
  .shrink:hover{
    -webkit-transform: scale(0.0,0.0);
    -moz-transform: scale(0.0,0.0);
    transform: scale(0.0,0.0);
  }
  .squezze:hover{
   -webkit-transform: scale(1.5, 0.0);
   -moz-transform: scale(1.5, 0.0);
   transform: scale(1.5, 0.0);
  }
  .deform:hover{
    -webkit-transform: skew(45deg,45deg);
    -moz-transform: skew(45deg,45deg);
    transform: skew(45deg,45deg);
  }
  </style>
  </head>
  <body>
    <div id="container">
    <h1>CSS3 Animated Buttons</h1>
    <button class="highlight">高亮</button>
    <button class="fade">淡出</button>
    <button class="boom">放大</button>
    <button class="focus">聚焦</button>
    <button class="rotate">旋转</button>
```

```
  <button class="knit">展开</button>
  <button class="shrink">收缩</button>
  <button class="squezze">挤压</button>
  <button class="deform">变形</button>
</div>
<script>
</script>
</body>
</html>
```

2.1.7　CSS3 动画

动画由一系列关键帧(key frames)组成，在 CSS3 中我们也可以创建多个关键帧，CSS3 使用@keyframes 指令创建动画效果。在@keyframes 中规定某项 CSS 样式，就能创建由当前样式逐渐改为新样式的动画效果。例如：

```
@keyframes mymove {
    0%   {top: 0px;}
    25%  {top: 200px;}
    75%  {top: 50px}
    100% {top: 100px;}
}
```

在上面的代码中，我们可以使用 from 替换 0%，使用 to 替换 100%，动画效果是一样的。只有把动画应用到某个元素时才会产生动画效果。我们至少需要指定动画的名称和时间。例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>CSS3</title>
<style>
div {
    width: 100px;
    height: 100px;
    background: red;
    position: relative;
    -webkit-animation: mymove 3s infinite; /* Chrome, Safari, Opera */
    animation: mymove 3s infinite;
}
/* Chrome, Safari, Opera */
@-webkit-keyframes mymove {
    0%   {top: 0px;}
    25%  {top: 200px;}
    75%  {top: 50px}
    100% {top: 100px;}
}
@-moz-keyframes mymove {
```

```
    0%   {top: 0px;}
    25%  {top: 200px;}
    75%  {top: 50px}
    100% {top: 100px;}
}
/* 标准语法 */
@keyframes mymove {
    0%   {top: 0px;}
    25%  {top: 200px;}
    75%  {top: 50px}
    100% {top: 100px;}
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

上面的代码运行后，div 元素会执行动画效果，div 元素的高度会在第 1 秒滑动到 200 像素位置，在第 2 秒滑动到 50 像素位置，在第 3 秒滑动到 100 像素位置。我们还可以指定其他的动画属性，如表 2-4 所示。

表 2-4　动画属性

属　性	描　述
animation-name	规定@keyframes 动画的名称
animation-duration	规定动画完成一个周期所花费的时间(单位为秒或毫秒)，默认是 0
animation-timing-function	规定动画的速度曲线，默认是 ease
animation-delay	规定动画何时开始，默认是 0
animation-iteration-count	规定动画被播放的次数，默认是 1
animation-direction	规定动画是否在下一周期逆向播放，默认是 normal。alernate 值表示动画应该轮流反向播放
animation-play-state	规定动画是否正在运行或暂停，默认是 running

2.2　less 简介

CSS 作为一门标记语言，语法相对简单，对使用者的要求较低，但同时也带来一些问题：CSS 需要书写大量看似没有逻辑的代码，不方便维护及扩展，不利于复用，尤其对于非前端开发工程师来讲，往往会因为缺少 CSS 编写经验而很难写出组织良好且易于维护的 CSS 代码，造成这些困难的很大原因源于 CSS 是一门非程序式语言，没有变量、函数、作用域等概念。于是就出现了 CSS 预处理技术，通俗来说，CSS 预处理器用一种专门的编程语言，先进行 Web 页面样式设计，然后编译成正常的 CSS 文件，以供项目使用。同时也涌现出了

很多种不同的 CSS 预处理器语言，比如 Sass(SCSS)、less 等。这里我们以 less 为例。

下面我们看一个 less 的简单例子(styles.less 代码)：

```
@color: #4D926F;
    h2 {
    background-color: @color;
}
```

经过编译生成的 CSS 文件如下：

```
h2 {
 background-color: #4D926F;
}
```

2.2.1　在客户端使用 less

我们可以直接在客户端使用.less 文件(less 源文件)，只需要从 http://lesscss.org 下载 less.js 文件或者使用 CDN 上的 less.js，并且在引用 less 文件之后引用 less.js 文件，例如：

```
<!DOCTYPE html>
<html>
<head lang="en">
<meta charset="UTF-8">
<title>less</title>
<link rel="stylesheet/less" type="text/css" href="styles.less">
<script src="//cdnjs.cloudflare.com/ajax/libs/less.js/2.6.1/less.min.js">
</script>
</head>
<body>
   <header>
      <h1>前端开发必备</h1>
   </header>
   <h2>less 教程</h2>
</body>
</html>
```

运行上面的代码，效果如图 2-11 所示。

图 2-11　第一个 less 例子

注意：要把代码部署到 Tomcat 服务器上才能正确显示，不然会出现 XMLHttpRequest cannot load 错误。建议使用 IntelliJ IDEA 14.0.3 创建 Static Web 项目，如图 2-12 所示。

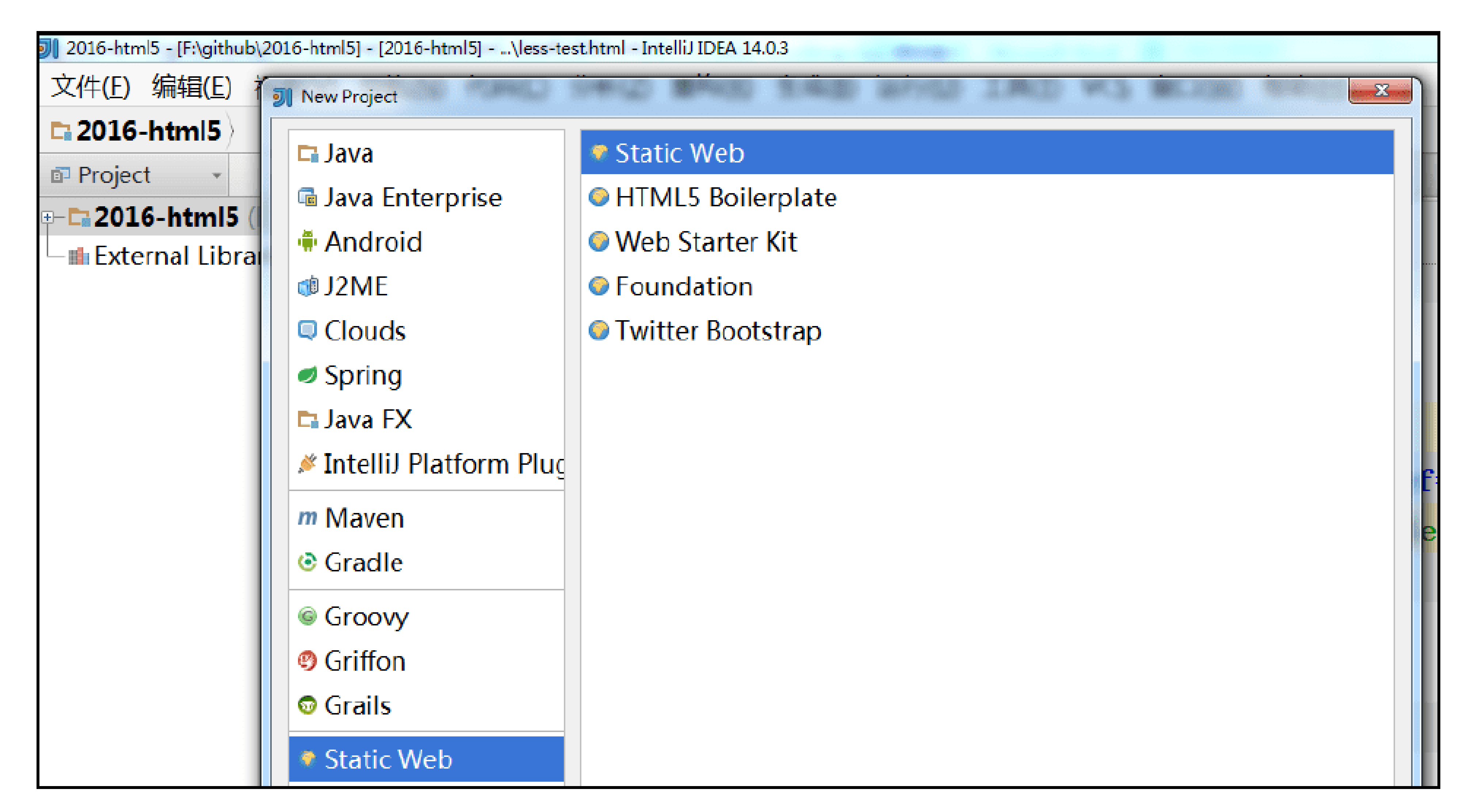


图 2-12　使用 IntelliJ IDEA 创建 Static Web 项目

右击代码区域，可以在浏览器中查看页面，如图 2-13 所示。

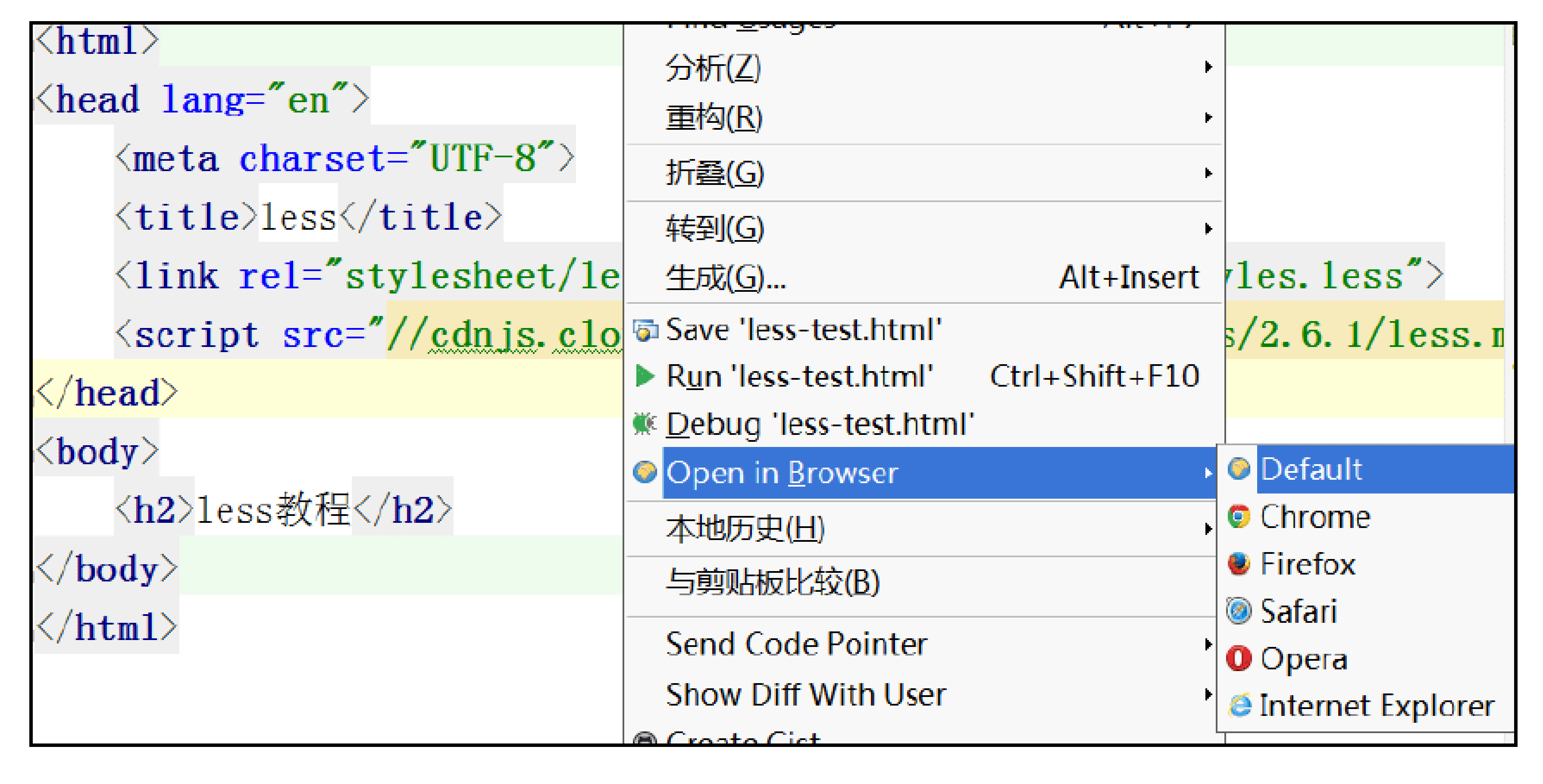


图 2-13　选择在浏览器中查看页面

下面我们学习一些基础用法，更多用法可以查看官方教程。

2.2.2　混合

混合可以将一个定义好的 class A 轻松引入到另一个 class B 中，从而简单实现 class B 继承 class A 中的所有属性。我们还可以带参数地进行调用，就像使用函数一样。

```
.rounded-corners (@radius: 5px) {
  border-radius: @radius;
  -webkit-border-radius: @radius;
  -moz-border-radius: @radius;
}

header {
  .rounded-corners;
}
```

```
footer {
  .rounded-corners(10px);
}
/* 生成的 CSS */
header {
  border-radius: 5px;
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
}
footer {
  border-radius: 10px;
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
}
```

2.2.3　嵌套规则

我们可以在一个选择器中嵌套另一个选择器来实现继承，这在很大程度上减少了代码量，并且代码看起来更加清晰。

```
header {
  h1 {
    font-size: 26px;
    font-weight: bold;
  }
  p { font-size: 12px;
    a { text-decoration: none;
      &:hover { border-width: 1px }
    }
  }
}
/* 生成的 CSS */
header h1 {
  font-size: 26px;
  font-weight: bold;
}
header p {
  font-size: 12px;
}
header p a {
  text-decoration: none;
}
header p a:hover {
  border-width: 1px;
}
```

2.2.4　函数&运算

运算提供了加、减、乘、除操作；我们可以做属性值和颜色的运算，这样就可以实现属性值之间的复杂关系。less 中的函数一一映射了 JavaScript 代码，如果愿意的话可以操作属性值。

```
/* less */
@the-border: 1px;
@base-color: #111;
@red: #842210;

header {
  color: @base-color * 3;
  border-left: @the-border;
  border-right: @the-border * 2;
}
footer {
  color: @base-color + #003300;
  border-color: desaturate(@red, 10%);
}
/* 生成的 CSS */
header {
  color: #333;
  border-left: 1px;
  border-right: 2px;
}
footer {
  color: #114411;
  border-color: #7d2717;
}
```

除了在客户端使用 less，在服务器端也可以使用 less。less 在服务器端的使用主要是借助于 less 的编译器，将 less 源文件编译生成最终的 CSS 文件。目前常用的方式是利用 node 的包管理器(npm)安装 less，安装成功后就可以在 node 环境中对 less 源文件进行编译。推荐的做法是将 less 文件编译生成静态 CSS 文件，然后在 HTML 文档中使用。

2.3　BootStrap 简介

BootStrap 是 Twitter 推出的一个开源的用于前端开发的工具包。它由 Twitter 的设计师 Mark Otto 和 Jacob Thornton 合作开发，是一个 CSS/HTML 框架。BootStrap 提供了优雅的 HTML 和 CSS 规范，由动态 CSS 语言 less 写成。BootStrap 一经推出就颇受欢迎。

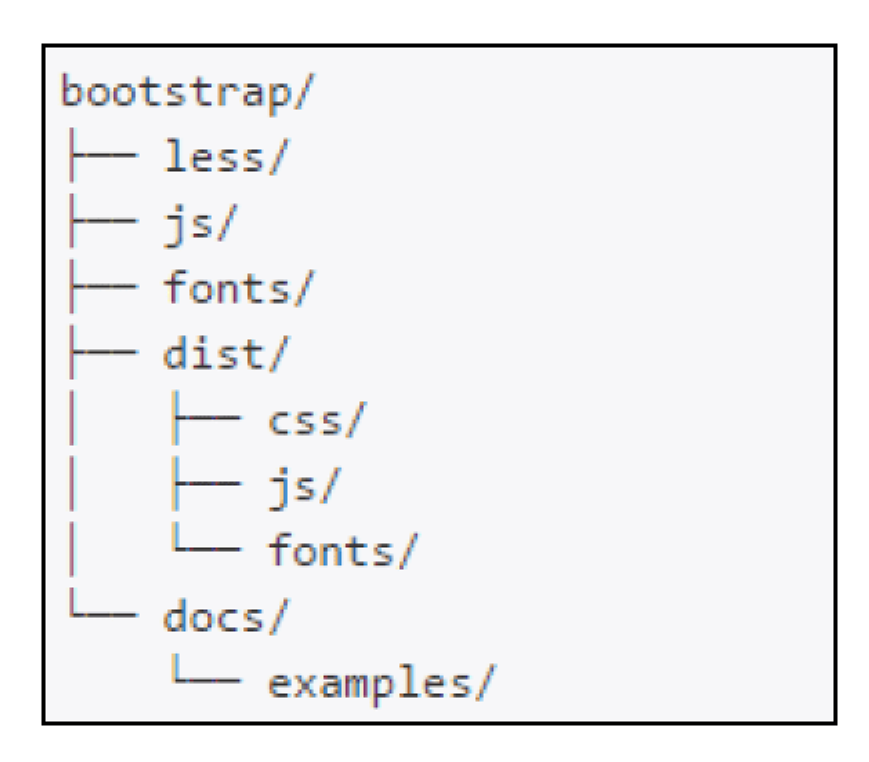
我们可以从 bootcss.com 下载 BootStrap，也可以应用 CDN 上的文件。

任务一：下载 BootStrap 并查看其目录结构

分别下载 BootStrap 源码包，以及编译后的用于生产环境的压缩包，然后查看其目录结构。

任务总结：

BootStrap 源码包的目录结构如图 2-14 所示。



```
bootstrap/
├── less/
├── js/
├── fonts/
├── dist/
│   ├── css/
│   ├── js/
│   └── fonts/
└── docs/
    └── examples/
```

图 2-14　BootStrap 源码包的目录结构

编译后的代码目录结构如图 2-15 所示。

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

图 2-15　编译效果

任务二：个人简历的制作

任务需求描述：

在北京，平均一个职位会收到 1000 封求职信，其中 200 封是合格的。据某网站统计，规模较大的企业一般每周要接收 500 份至 1000 份电子简历，其中的 80%在管理者浏览不到 30 秒后就被删除了。要让别人在半分钟内通过一份 e-mail 对你产生兴趣，其难度与跟用人单位直接见面相比难得多，因此一份简历对于一名求职者而言意义重大。下面这个任务就是要制作一份自己的 HTML5 个人简历，如图 2-16 所示。



图 2-16　个人简历

任务要求：

1) 使用 BootStrap 完成界面的布局

2) 使用 less 优化自定义的 CSS，这里的文件名为 styles.css

预备知识：

1) BootStrap 中用到一些 HTML 元素和 CSS 属性，需要将页面设置为 HTML5 文档类型，即：

```
<!DOCTYPE html> <html lang="zh-CN"> ... </html>
```

2) 布局容器：BootStrap 需要为页面内容和栅格系统包裹一个 .container 或 container-fluid(占据全部视口(view port)的容器)容器。

```
<div class="container"> ... </div>
```

3) 栅格系统：BootStrap 提供了一套最多 12 列的流式栅格系统，通过 .row 表示行和 .col-xs-4 等表示宽度的列。

栅格系统中的列通过指定 1 到 12 的值来表示其跨越的范围。例如，三个等宽的列可以使用三个 .col-xs-4 来创建，具体如图 2-17 所示。

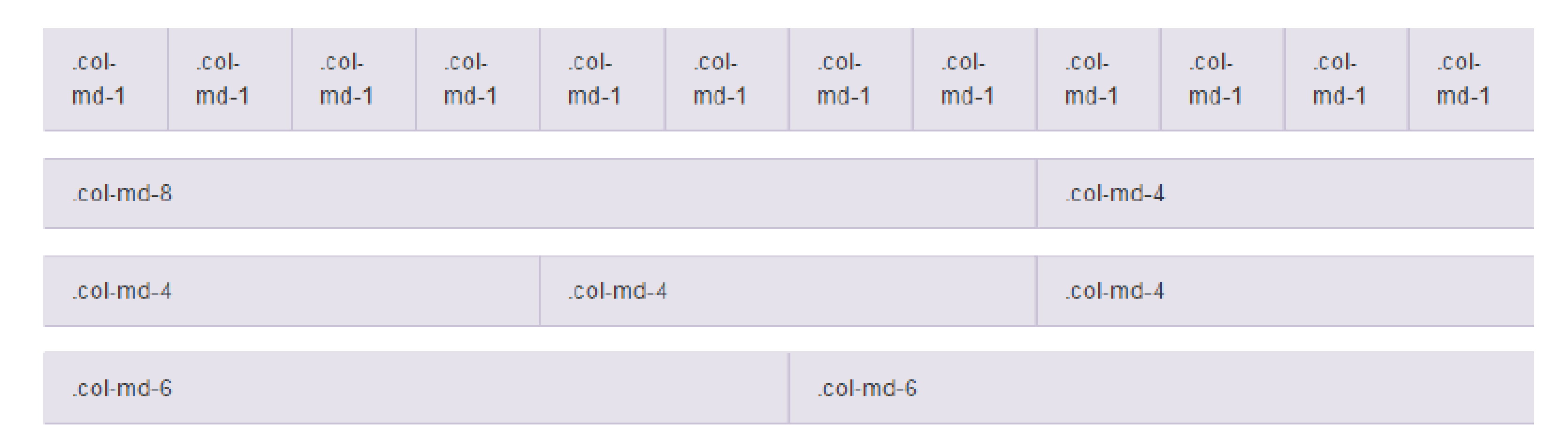


图 2-17　栅格系统

对应的代码为：

```
<div class="row">
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
<div class="col-md-1">.col-md-1</div>
</div>
<div class="row">
<div class="col-md-8">.col-md-8</div>
<div class="col-md-4">.col-md-4</div> </div>
<div class="row">
<div class="col-md-4">.col-md-4</div>
<div class="col-md-4">.col-md-4</div>
<div class="col-md-4">.col-md-4</div>
</div>
<div class="row">
<div class="col-md-6">.col-md-6</div>
<div class="col-md-6">.col-md-6</div>
</div>
```

BootStrap 的栅格系统可以在多种屏幕设备上工作，能通过类前缀.col-xs-、.col-sm-、.col-md-、.col-lg-来标记针对不同尺寸屏幕的列宽度，具体见表 2-5。

表 2-5　多屏幕支持参数

	超小屏幕 手机(<768px)	小屏幕 平板电脑 (≥768px)	中等屏幕 桌面显示器 (≥992px)	大屏幕 大桌面显示器 (≥1200px)
栅格系统行为	总是水平排列	开始是堆叠在一起的，当大于这些阈值时将变为水平排列		
.container 最大宽度	(自动)	750px	970px	1170px
类前缀	.col-xs-	.col-sm-	.col-md-	.col-lg-

这里给出完成项目需要的图片资源和 CSS 样式表，项目结构如图 2-18 所示。

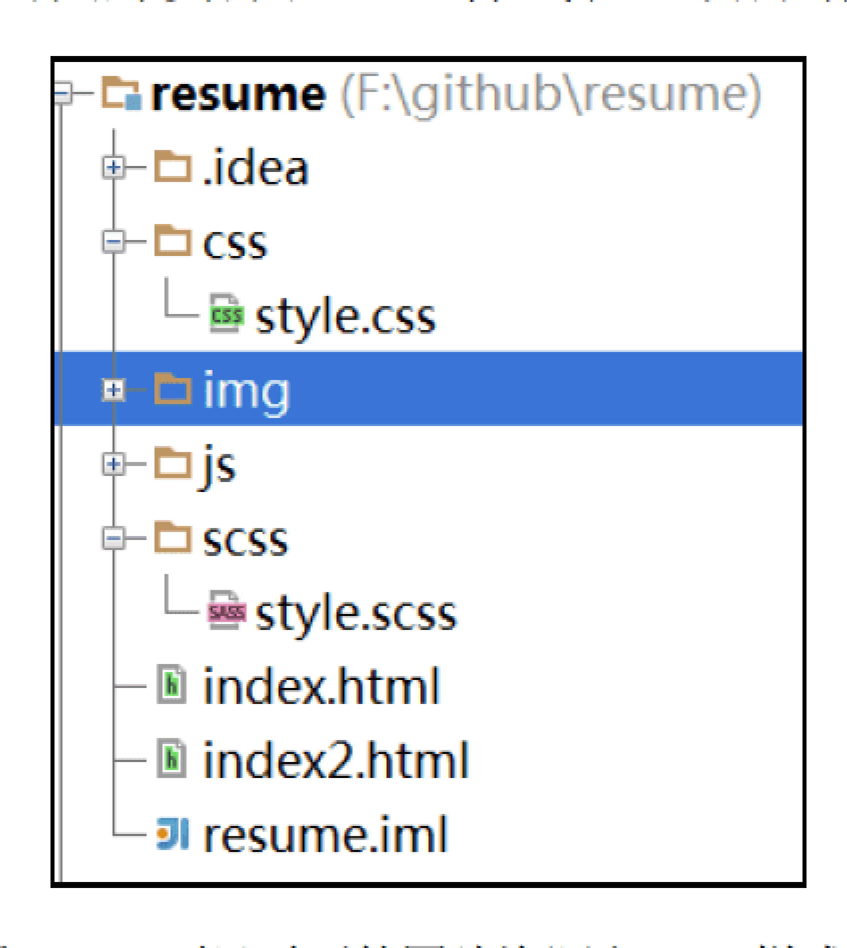


图 2-18　项目需要的图片资源和 CSS 样式表

style.css 代码如下：

```
body {
  padding-bottom: 0;
  font-family: 'Open Sans', Arial, Tahoma;
  color: #363636;
  background: #ededed;
  font-size: 16px;
}
h2 {
  font-weight: 700;
  text-transform: uppercase;
  line-height: 1.4em;
  font-size: 1.5em;
}
.primary,
strong,
h2 {
  color: #3b5a76;
}
/*设置容器 */
.container-main {
  background: #fff;
  margin-top: 80px;
```

```
  padding: 0px 30px 30px 30px;
}
.row {
  margin-bottom: 20px;
}
header {
  min-height: 100px;
  overflow: auto;
}
header h1 {
  margin: 0;
  padding: 20px 0 0 0;
  font-size: 2.2em;
  font-weight: 700px;
  text-transform: uppercase;
  letter-spacing: -1px;
  text-align: center;
  line-height: 1.4em;
}
/* 头像*/
.profile-img {
  width: 160px;
  display: block;
  margin: auto;
  margin-top: -80px;
}
.line {
  max-width: 100%;
}
/* 联系方式*/
.contact-list {
  padding: 0;
  margin: 0;
}
.contact-list li {
  list-style: none;
  font-weight: bold;
  line-height: 3.2em;
}
.contact-list span {
  display: block;
  margin: 0;
  padding: 0;
  line-height: 0.1em;
  margin-left: 37px;
  font-size: 15px;
  color: #ccc;
}
/* 教育$经验 */
.media {
  background: #ededed;
  padding: 20px;
```

```
}
.work-list li {
  list-style: none;
  float: left;
  padding: 0 10px 0 10px;
  margin-bottom: 30px;
}
/* 页脚 */
footer {
  background: #3b5a76;
  color: #fff;
  text-align: center;
  padding-top: 20px;
  padding-bottom: 20px;
}
```

实现步骤：

(1) 创建 HTML5 页面，设置其支持在移动设备浏览器上访问。引入 BootStrap，这里我们使用 CDN 上的 BootStrap 文件，并引入 style.css。创建基本的页面结构。

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>小明的求职简历</title>
    <!-- 新 BootStrap 核心 CSS 文件 -->
    <link rel="stylesheet"
href="http://cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min.css">
    <!-- jQuery 文件。务必在 bootstrap.min.js 之前引入 -->
    <script src="http://cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>
    <!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
      <script
src="http://cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script
>
    <link rel="stylesheet/less" type="text/css" href="css/style.css">
  </head>
  <body>
    <section>
    </section>
    <footer>
      <div class="container">
        <p>Copyright &copy; 2015, All Rights Reserved</p>
      </div>
    </footer>
  </body>
</html>
```

(2) 完成整个页面的布局设计，根据页面需求，我们知道主区域可以分为 4 行，第 1 行

是头像及姓名区域，第 2 行为“关于我”和“联系方式”(分两列，一列占 7/12 宽度，另一列占 5/12 宽度)，第 3 行为“教育经验”和“编程技能”，第 4 行为“求职意向”和“我的作品”。代码如下：

```
<section>
<div class="container container-main">
  <img src="img/person.png" class="profile-img">
  <header>
    <h1>您好 <br> 我的名字是 <strong>小明</strong> 我是一个前端工程师</h1>
  </header>
  <img src="img/line.png" class="line">
  <div class="row">
    <div class="col-md-7">
      <h2># 关于我</h2>
    </div>
    <div class="col-md-5">
          <h2># 联系方式</h2>
    </div>
  </div>
  <div class="row">
    <div class="col-md-7">
      <h2># 教育 & 经验</h2>

    </div>
    <div class="col-md-5">
      <h2># 编程技能</h2>
    </div>
  </div>

  <div class="row">
    <div class="col-md-7">
     <h2># 求职意向</h2>

   </div>
   <div class="col-md-5">
      <h2># 我的作品</h2>

   </div>
 </div>

</div>
</section>
```

(4) 实现联系方式 div。

(5) 实现“教育经验和编程技能”div。自学 BootStrap 进度条，包括 progress progress-bar progress-bar-success progress-bar-striped，完成编程技能条的效果。

(6) 实现“求职意向”和“我的作品”div。自学 BootStrap 警告框(Alert)用法，包括 alert-success、alert-info 等，完成“求职意向”div。

(7) 使用 scss 优化 CSS 代码，在 scss 文件夹下创建 style.scss 文件，提取 CSS 文件中 body 区域的字体大小、字体颜色、背景颜色、标题和 footer 的主要颜色为变量，并将修改后的内容保存到 style.scss。在 head 的最后引用 style.scss 和 less.min.js。

本章小结

本章介绍了 CSS3 关于动画的新特性。CSS3 增加了两种动画实现方式：一种是 transition(过渡)，另一种是 animation(动画)。CSS3 动画比 jQuery 动画更轻量，性能更好，更易于实现，同时也不必担心缺乏标准。最后又学习了 BootStrap 的入门知识。

本章练习

1. CSS3 中 transition(过渡)和 animation(动画)的区别是什么？
2. less 的作用是什么？

第3章　JavaScript

JavaScript 是一种广泛使用的编程语言，从前端网页到移动 Web 应用程序，JavaScript 无处不在。根据编程语言社区排行榜的数据，JavaScript 的热门度越来越高。一方面是因为 JavaScript 比较容易学习，另一方面是近几年 HTML5 的出现，将 JavaScript 提升到了前所未有的高度。它既适合当作学习编程的入门语言，也适合当作日常开发的工作语言。它是目前最有希望、前途最光明的计算机语言之一。

本章内容：

- 如何使用 JavaScript 变量、循环遍历、内置对象
- 如何使用 JavaScript 正则表达式
- 如何使用 JavaScript 对象和函数
- 文档对象模型(DOM)
- 浏览器事件模型

3.1　JavaScript 简介

3.1.1　JavaScript 是什么

当互联网用户还仅仅通过 28.8 Kb/s 的调制解调器连接到网络时，设想一下，用户填完一个表单，单击“提交”按钮，等待了 30 秒的处理后，看到的却是一条告诉用户“你忘记填写一个必要的字段”的消息。这会让用户多么痛苦。随着网络的普及，对于开发客户端脚本的需求也逐渐增大。1995 年，Netscape(网景)公司的 Brendan Eich 在网景导航者浏览器上首次设计实现了 LiveScript 脚本语言。因为网景公司正在与 Sun 公司合作，网景公司的管理层希望它外观看起来像 Java，因此取名为 JavaScript。微软为了获取技术优势，决定进军浏览器，发布了 IE 3.0 并搭载了一个 JavaScript 的克隆版，叫做 JScript。1997 年，JavaScript 1.1 作为一个草案提交给欧洲计算机制造商协会(ECMA)。第 39 技术委员会(TC39)被委派来“标准化一门通用、跨平台、中立于厂商的脚本语言的语法和语义”，由来自 Netscape、Sun、微软、Borland 和其他一些对脚本编程感兴趣的公司的程序员组成的 TC39 推出了 ECMA-262，该标准定义了名为 ECMAScript 的全新脚本语言。以后的 JavaScript 遵循了 ECMAScript 标准，并且还加入了 DOM(文档对象模型)和 BOM(浏览器对象模型)。

本书中使用的 JavaScript 版本为 ECMAScript 5。

JavaScript 是一种解释型脚本语言，是一种动态类型、弱类型的语言。它的解释器被称为

JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言。它最早是在 HTML 网页上使用，用来给 HTML 网页增加动态功能。其次，JavaScript 是基于对象的语言，基于对象的语言含有面向对象语言的思想，但比面向对象语言简单。再次，JavaScript 是基于事件驱动的语言，它是在用户界面的环境下，使得一切输入变化简单化。我们通常将鼠标或热键的动作称为事件(Event)，而将由鼠标或热键引发的一连串程序的动作称为事件驱动(Event Driver)，对事件的响应由编写的事件响应处理函数来完成。JavaScript 也是浏览器的编程语言，只要有浏览器存在就能够解释 JavaScript，与操作系统无关，因此它具有跨平台性。

在人人微博网站中，我们希望用户的登录名既可以是邮箱地址也可以是手机号，这就需要使用 JavaScript 在客户端进行验证。还希望在首页有一个公告栏，能够滚动显示一些信息，如图 3-1 所示。

滚动文字消息　　最扯分手理由有哪些　　第三届微小说大赛

图 3-1　滚动显示

这些实用的功能，在学习本章知识的过程中，本书会逐一实现。

3.1.2　JavaScript 的作用

- **富客户端开发(RIA)：** gmail、google reader 等都是 RIA 的典型。使用 JavaScript 可以验证表单字段、动态地更改页面元素的外观、滚动内容等。Firefox 的界面就是用 JavaScript 实现的，我们可以到 Firefox 的安装目录下随便解压一个 jar 包看看，里面有大量的 JavaScript 代码。
- **浏览器的平台化：** 随着 HTML5 的出现，浏览器本身的功能越来越强，不再仅仅能浏览网页，而是越来越像一个平台，JavaScript 因此得以调用许多系统功能，比如操作本地文件、操作图片、调用摄像头和麦克风等。这使得 JavaScript 可以完成许多以前无法想象的事情。
- **服务器端：** Node.js 项目使得 JavaScript 可以用于开发服务器端的大型项目，网站的前后端都用 JavaScript 开发已经成为现实。
- **数据库操作：** JavaScript 甚至也可以用来操作数据库。NoSQL 数据库(将在第 5 章介绍)这个概念，本身就是在 JSON(JavaScript Object Notation，JavaScript 对象表示法)格式的基础上诞生的，大部分 NoSQL 数据库允许 JavaScript 直接操作。基于 SQL 语言的开源数据库 PostgreSQL 支持 JavaScript 作为操作语言，可以部分取代 SQL 查询语言。

3.1.3　在网页中插入 JavaScript 的方法

我们先来看看第一个 JavaScript 脚本，代码如下：

```
<!DOCTYPE html>
<html>
    <head>
```

```
    <title>简单的 JavaScript Hello World</title>
        <script>
            document.write("Hello, world!");   // 在浏览器视窗内直接显示
         </script>
    </head>
    <body>
          HTML 内容……
    </body>
</html>
```

在本例中，把<script>元素置于网页的 head 部分，然后在其中加入脚本程序。脚本中使用 write 方法向网页中添加了一个新文本行。图 3-2 展示了这一简单页面的效果。

```
Hello, world! HTML 内容……
```

图 3-2　页面的效果

JavaScript 在页面中放置的位置很重要。<script>元素也可以位于 body 中，脚本语言的执行顺序与书写顺序相同。在浏览器解析 HTML 文档时，无论何时遇到脚本块，都会停止对 HTML 代码的解析，转而执行脚本块中的代码。示例如下：

```
<!DOCTYPE html>
<html>
    <head>
    <title>简单的 JavaScript Hello World</title>
        <script>
            document.write("Hello, world!");   // 在浏览器视窗内直接显示
         </script>
    </head>
    <body>
          HTML 内容……<br>
        <script>
            document.write("Hello, world 123!");   // 在浏览器视窗内直接显示
         </script>
    </body>
</html>
```

代码效果如图 3-3 所示。

```
Hello, world! HTML 内容……
Hello, world 123!
```

图 3-3　代码效果

但一般不推荐这样使用，这样对于页面的性能和可维护性都是不好的做法。另一种添加 JavaScript 的方法是引入文件扩展名为.js 的外部文档。如果脚本用于多于一个页面的话，就不需要在每个使用该脚本的页面中重复其内容。如果需要更新脚本，则只需要修改一处的内容。同时这样也使得整个 HTML 页面更加简洁和易读。只要使用<script>元素的 src 属性指向包含 JavaScript 文件的绝对或相对 URL 即可，比如：

```
<script src="scripts/changeinfo.js"></script>
```

3.1.4　如何调试 JavaScript

所有的主流浏览器现在都带有一个运行 JavaScript 的接口，叫做控制台。JavaScript 程序开发人员可以在控制台中记录应用程序的日志信息，并可以直接运行 JavaScript 代码，这样可以输入一个或一系列命令，然后立刻看到结果。我们也可以交互式对 JavaScript 语言的基础代码进行演练。可以在 Google Chrome 浏览器中找到开发者工具栏，也可以使用快捷键 Control+Shift+J 打开控制台，如图 3-4 所示。



图 3-4　JavaScript 控制台

我们在图 3-4 所示的控制台中输入了 console.log("Web 编程")，并得到了输出结果。这样可以工作的原因是控制台使用了拥有方法 log()的 console 对象，使我们能够向控制台写入消息。

在 Sources 标签栏中可以调试 JavaScript，如图 3-5 所示。

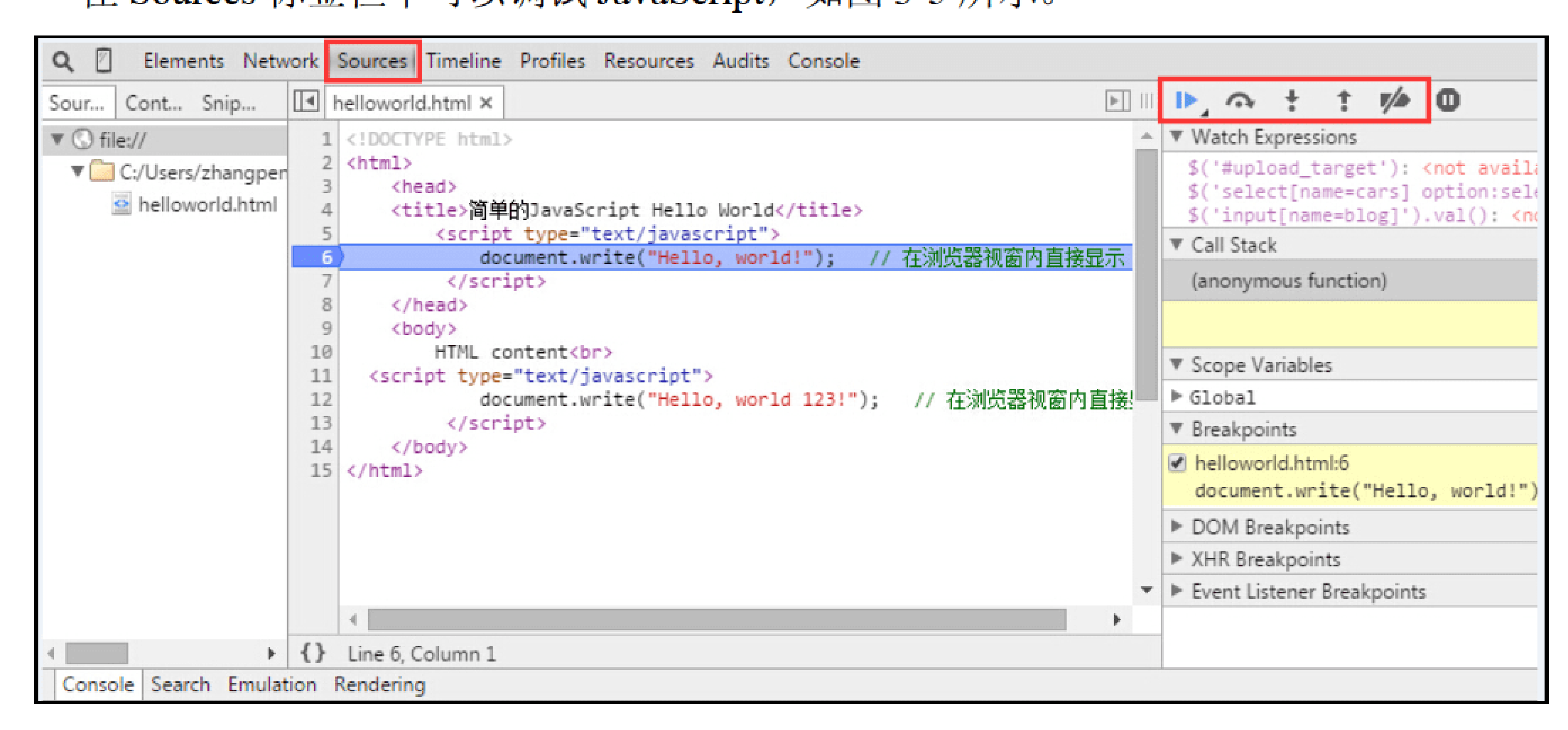


图 3-5　调试 JavaScript

在 Firefox 中还有同样强大的 Firebug 插件，它集 HTML 查看和编辑、JavaScript 控制台、网络状况监视器于一体，是开发 JavaScript、CSS、HTML 和 Ajax 的得力助手。

下载地址是 https://getfirebug.com/。可以使用快捷键 F12 打开 Firebug，如图 3-6 所示。

图 3-6　Firebug

为了容易阅读代码和方便调试，我们可以在代码中加入注释。可以通过两种方式在 JavaScript 代码中添加注释。第一种是使用双正斜线字符，例如：

```
<script>
          document.write("Hello, world!");   // 在浏览器视窗内直接显示
 </script>
```

第二种是将注释内容放置于起始符号/*与结束符号*/之间，例如：

```
/*声明
   变量*/
var a = "js";
```

3.2　JavaScript 基础语法

3.2.1　JavaScript 的变量

```
var name = "john";
age = 20;
```

JavaScript 声明变量时使用 var，因为 JavaScript 采用弱类型的变量形式，声明时不需要变量类型，这与 Java 语言不同，在赋值时自动确定变量的类型。var 可以省略，但是不建议省略。另外，变量的名称必须遵循一些规则：变量名区分大小写；首字母必须是字母、下划线(_)或美元符号($)；余下的字母可以是下划线、美元符号、任意字母或数字；变量名中不能

有空格、回车符或其他标点字符；变量名不能是关键字(如表 3-1 所示)。

表 3-1　JavaScript 关键字

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	instanceof	throw	while
debugger	finally	new	true	with
default	for	null	try	

另外 JavaScript 还有一些保留字(如表 3-2 所示)，这些保留字以后 JavaScript 可能会使用。

表 3-2　JavaScript 保留字

abstract	double	goto	native	static
boolean	enum	implements	package	super
byte	export	import	private	synchronized
char	extends	int	protected	throws
class	final	interface	public	transient
const	float	long	short	volatile

如果只是声明了变量而没有使用，例如：

```
var school;
```

这时变量 school 的值为 undefined，可以理解为是一个常量，用来表示变量没有赋值。下面我们看一下给变量赋值：

```
name = "peter";
age = "90s";
```

3.2.2　JavaScript 的 6 种数据类型

JavaScript 的数据类型分为两类：原始类型(primitive type)和对象类型(object type)。原始类型包括数值(number)、字符串(string)、布尔值(boolean)、空(null)和未定义(undefined)。其中 null 和 undefined 是特殊类型，以后会详细介绍。对象类型只有对象(object)，这是因为函数、数组等都被视为对象。

1. 字符串

字符串由零个或多个字符构成。字符可以是字母、数字、标点符号或空格。字符串必须放在单引号或双引号中。字符串常量必须使用单引号或双引号引起来，如果一个字符串中含有双引号，则只能将该字符串放在单引号中。例如：

```
var tmp = ' "OK",I said. '; //输出"OK",I said.
```

更通用的方法是使用转义字符(escaping) “\” 实现特殊字符按原样输出。例如：

```
var tmp = " Consumer Rights Day 3\'  15" ;// 输出 Consumer Rights Day 3'  15
```

2. 数值型

JavaScript 提供了统一的数值表示方法，它不区分整型和浮点型。数值型数据和字符型数据的区别是数值型数据不用引号引起来。例如，下面都是正确的数值表示法：

```
var num1 = 37;
var num2 = 37.32;
var num3 = -3e7;
```

3. 布尔型

布尔型数据的值只包含 true 和 false。布尔型数据不能用引号引起来，否则就变成字符串了。例如：

```
var approved = true;
```

4. null

null 值用来指出对象属性不包含值。通常，如果设置一个属性包含值，但出于某种原因这个值还不能用，那么应该使用 null 值来表明属性拥有一个空值。例如：

```
var myObject = new Object();
myObject.name=null;
console.log(myObject.name);// 输出 null
```

那么如何判断一个值是不是 null 呢？对 null 使用 typeof 运算符会返回 object，显然使用 typeof 不能判断一个值是不是 null。为了更明确地验证 null 值，应该使用===(强等于)运算符。例如：

```
var myObject = null;
console.log(typeof  myObject);//输出 object
console.log(myObject === null);//输出 true
```

5. undefined

有两种情况会用到 undefined：一种是声明了变量(如 var foo)而没有指定值，第二种情况是试图访问的对象属性没有被定义。例如：

```
var initializedVariable; // 声明变量
console.log(initializedVariable); // 输出 undefined
console.log(typeof initializedVariable); // 输出 undefined
var foo = new Object();
console.log(foo.bar); // 输出 undefined
console.log(typeof  foo.bar); // 输出 undefined
```

6. 对象包装

JavaScript 使用“字面量”的快捷方式创建大多数原始类型，而 JavaScript 对象则可以使用 new 操作符创建，比如：

```
var obj = new Object();
obj.name='zs';
```

JavaScript 对象会在下节详细介绍，由于在实际使用中对 string、number、boolean 的操作非常多，JavaScript 进行了进一步的包装，提供了内置的 String()、Number()、Boolean()对象。我们可以使用 new 操作符创建 String 对象、Number 对象、Boolean 对象。 例如：

```
var stringObject = new String('foo');
var numberObject = new Number(1);
var booleanObject = new Boolean(false);
```

我们可以使用 typeof 运算符返回数据的类型，例如：

```
// string, number, and boolean 对象
console.log(typeof  new String('foo')); // 输出 'object'
console.log(typeof  new Number(1)); //输出'object'
console.log(typeof  new Boolean(true)); //输出'object'

// string, number, and boolean 字面量/原始值
console.log(typeof  'foo'); //输出'string'
console.log(typeof  1); //输出'number'
console.log(typeof  true); //输出'boolean'
```

字符串对象实例的常用属性有 length 属性，常用的方法如表 3-3 所示。

表 3-3　JavaScript 字符串常用的方法

方　法	说　明	示　例

charAt	返回一个字符串的给定位置的字符，位置从 0 开始编号	`var s = new String("abc");`
		`s.charAt(1)//"b"`
		`s.charAt(s.length - 1)  // "c"`

charCodeAt	返回给定位置字符的 Unicode 编码	`var s = new String("abc");` `s.charCodeAt(1)//  98`

concat	用于连接两个字符串	`var s1 = "abc";`
		`var s2 = "def";`
		`s1.concat(s2)  //  "abcdef"`

substring	用来返回一个字符串的子串，第一个参数表示子字符串的开始位置，第二个参数表示结束位置	`var a = 'The Three Musketeers';` `a.substring(4, 9)  //  'Three'`
substr	用来返回一个字符串的子串，第一个参数是子字符串的开始位置，第二个参数是子字符串的长度	`var b = 'The Three Musketeers';` `b.substr(4, 9)  //  'Three Mus'`
slice	用来返回一个字符串的子串，第一个参数是子字符串的开始位置，第二个参数是子字符串的结束位置	`var c = 'The Three Musketeers';` `c.slice(4, 9)  //  'Three'`
indexOf	用于确定一个字符串在另一个字符串中的位置，从字符串头部开始匹配。如果返回-1，就表示不匹配	`"hello world".indexOf("o");`

(续表)

方　法	说　明	示　例
lastIndexOf	用于确定一个字符串在另一个字符串中的位置，从尾部开始匹配	"hello world".lastIndexOf("o") // 7
trim	用于去除字符串两端的空格	" hello world ".trim() // "hello world"
toLowerCase	用于将一个字符串转为小写	"Hello World".toLowerCase() // "hello world"
toUpperCase	用于将一个字符串转为大写	"Hello World".toUpperCase() // "HELLO WORLD"
match	匹配字符串，返回一个数组，返回的数组拥有 index 属性和 input 属性，分别表示匹配字符串开始的位置(从 0 开始)和原始字符串	var matches = "cat, bat, sat, fat".match("at"); matches // ["at"] matches.index // 1 matches.input // "cat, bat, sat, fat"
search	等同于 match，但是返回值为匹配的第一个位置	"cat, bat, sat, fat".search("at") // 1
replace	替换匹配的子字符串	"aaa".replace("a", "b") // "baa"
split	按照给定规则分割字符串，返回一个由分割出来的各部分组成的新数组	"a|b|c".split("|") // ["a", "b", "c"]

3.2.3　JavaScript 的运算符

JavaScript 的运算符和 Java 的语法相似，包括加、减、乘、除，自增(++)、自减(--)、赋值运算符、比较运算符、逻辑运算符、位运算符等。例如：

```
var a = 1;
a = a+1;//a 的值为 2
a += 1; //a 的值为 3
a++;//a 的值为 4
++a;; //a 的值为 5
```

使用加号、减号的时候，如果两边的数据类型不一致，会使用隐式类型转换。例如：

```
"18"-2 //输出 16
"18"+2 //输出 182
```

我们可以巧用这个规则进行数据类型转换，比如把 string 转换为 number，只要减 0 就可以了，把 number 转换为 string 只要加空字符串就可以了。代码如下：

```
var  str = "10";
var num = str -0;
console.log(typeof num);// 输出 number
var s = num+"";
console.log(typeof s);// 输出 string
```

比较运算符用来比较两个操作数的大小。实例见表 3-4。

表 3-4　JavaScript 比较运算符

运 算 符	说　　明	示　　例
==	是否相等(只检查值)	a=7,b="5"; a==b;//true
===	是否全等(检查值和数据类型)	a=7,b="5"; a===b// false
!=	是否不等于	a=7,b="7"; a!=b;//false
!==	是否不全等于	a=7,b="7"; a!==b;//true
>、<、>=、<=	大于、小于、大于等于、小于等于	a=7,b=5; a>b;//true

使用==进行类型比较时，如果两边的数据类型不一致，也会进行隐式类型转换，然后再比较。例如：

```
1=="1.0"//输出 true
```

上面的例子中，string 和 number 比较会将 string 转换为 number，然后再比较：

```
1==true//输出 true
0==false//输出 true
2==true//输出 false
```

上面的例子中，boolean 和 number 比较会将 true 转换为 1、false 转换为 0，然后再比较。另外，对象只有在引用相同的对象(即有相同的地址)时才相等。在下面的示例中，objectFoo 和 objectBar 有相同的属性，但通过==或者===询问它们是否相等时，JavaScript 会告诉我们：它们不相等。

```
var objectFoo = new Object();
objectFoo.name = "name";
var objectBar = new Object();
objectBar.name = "name";
console.log(objectFoo == objectBar);// 输出 false
console.log(objectFoo === objectBar);// 输出 false

var objectA = new Object();
objectA.name = "name";
var objectB = objectA;
console.log(objectA == objectB);// 输出 true
console.log(objectA === objectB);// 输出 true
```

null 与 undefined 比较时有些特殊，用==比较两者是相等的。例如：

```
null == undefined //输出 true
null === undefined //输出 false
null == null//输出 true
undefined == undefined//输出 true
```

逻辑运算符包括&&(逻辑与)、||(逻辑或) 、!(逻辑非)、^(逻辑异或)。位运算符包括<<(左

移)、>>(右移)、~(取补)。此外还有其他的运算符：

- **new 运算符：** 用来创建一个对象或生成一个对象的实例。
- **typeof 运算符：** 返回一个用来表示表达式的数据类型的字符串。
- **in 运算符：** 测试一个对象中是否存在一种属性。例如：

```
var myObject = new Object();
myObject.name = "Jam";
myObject.age = "22";
myObject.phone = "555 5555";
if ("phone" in myObject)
   console.log ("property is present");
else
   console.log ("property is not present");
```

- **instanceof 运算符：** 返回一个布尔值，该值指示一个对象是否为特定类的一个实例。例如：

```
var obj = new Object();
obj.name='zs';
obj instanceof Object//输出 tue
```

- **条件运算符：** 这是 JavaScript 唯一的三目运算符，即它的操作数有 3 个。例如：

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

如果变量 visitor 中的值是 PRES，则向变量 greeting 赋值 Dear President，否则赋值 Dear。

3.2.4　JavaScript 流程控制

1. if 语句

if 语句的语法是：

```
if (condition) statement1 else statement2
```

或者

```
if (condition1) statement1 else if (condition2) statement2 else statement3
```

例如：

```
if (i > 10) {
  alert("大于 10");
} else if (i < 0) {
  alert("小于 0");
} else {
  alert("在 0 到 10 之间");
}
```

2. 循环语句

循环通常用于迭代数组的值，或者执行重复的算术任务。下面介绍 ECMAScript 提供的 4 种迭代语句。

while 语句是前测试循环。这意味着退出条件是在执行循环内部的代码之前计算的。因此，循环主体可能根本不被执行。它的语法如下：

```
while (expression) statement
```

例如：

```
var i = 0;
while (i < 10) {
  i += 2;
}
```

do-while 与 while 语句不同，它是后测试循环，即退出条件在执行循环内部的代码之后计算。这意味着在计算表达式之前，至少会执行循环主体一次。

它的语法如下：

```
do {statement} while (expression);
```

例如：

```
var i = 0;
do {i += 2;} while (i < 10);
```

for 语句是前测试循环，而且在进入循环之前，能够初始化变量，并定义循环后要执行的代码。

它的语法如下：

```
for (初始表达式; 循环条件表达式; 计数表达式) {语句块}
```

注意：计数表达式之后不能写分号，否则无法运行。例如：

```
iCount = 6;
for (var i = 0; i < iCount; i++) {
  alert(i);
}
```

这段代码定义了初始值为 0 的变量 i。只有当条件表达式(i < iCount)的值为 true 时，才进入 for 循环，这样循环主体可能不被执行。如果执行了循环主体，那么将执行循环后表达式，并迭代变量 i。

3. for-in 语句

for-in 语句用来枚举对象的属性。它的语法是：

```
for (property in expression) statement
```

例如：

```
var objectA = new Object();
objectA.name = "name";
for (sProp in ObjectA) {
  console.log(sProp);
}
```

4. break 和 continue 语句

break 和 continue 语句为循环中的代码提供了退出循环的方法。break 和 continue 语句的不同之处是 break 语句可以立即退出循环，阻止再次反复执行任何代码；而 continue 语句只是退出当前循环，根据控制表达式还允许进入下一次循环。例如：

```
var iNum = 0;
for (var i=1; i<10; i++) {
  if (i % 5 == 0) {
    break;
  }
  iNum++;
}
alert(iNum);//输出 "4"
```

如果用 continue 语句代替这个例子中的 break 语句，结果将不同：

```
var iNum = 0;
for (var i=1; i<10; i++) {
  if (i % 5 == 0) {
    continue;
  }
  iNum++;
}
alert(iNum);//输出 "8"
```

5. try-catch 语句

当 JavaScript 引擎执行 JavaScript 代码时，会发生各种错误：可能是语法错误，通常是程序员造成的编码错误或错别字；可能是拼写错误或语言中缺少的功能(可能缘于浏览器差异)。JavaScript 提供了异常捕获机制，语法是：

```
try
{
     //代码块
}
catch (exception){
     //处理错误
}
finally{
     //代码块
}
```

如果没有发生任何异常，catch 中的代码就会被忽略，但是不管有没有异常，最后都会执行 finally 中的代码。例如：

```
try{
    alertE("hello");
}catch(err)
{
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.message + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
}
```

我们故意把 alert 错写为 alertE，catch 块会捕捉到 try 块中的错误，并执行代码来处理它。try 语句块的后面可以有一个 catch 语句块或 finally 语句块，当然也可以两者都有。例如：

```
try{
  alert("hello");
}finally{
  console.log("finally");
}
```

如果要创建自定义错误，可使用 throw 语句。例如：

```
try{
    try{
        throw "test";
    }
    finally{
    console.log("finally");
    }
}
catch (ex) {console.log(ex);}
```

在上面的例子中，我们嵌套使用了 try-catch 语句，内层的 try 语句抛出了自定义错误，但是没有 catch 语句，在程序跳出到外层时会先执行内层的 finally 语句，所以先输出 finally，再输出 test。

6. block 语句(块语句)

块语句常用于组合零到多个语句。块语句用一对花括号定义。语法是：

```
{
    语句 1
    语句 2
    语句 n
}
```

块语句通常和 if、while、for 等一起使用，使用 JavaScript 块语句需要注意的一点是：没有块级作用域。例如：

```
for(var i = 0;i < 5;i++){
    var str = "ok";
    console.log(str);
}
```

这段常用的 for 循环块给人的感觉是变量 i 只能在 for 循环中访问。但事实上由于没有块级作用域，在 for 循环块外也可以访问变量 i。上面的代码与下面的代码是等价的：

```
var i = 0
for(;i < 5;i++){
    var str = "ok";
    console.log(str);
}
console.log(i);//输出 5
```

后面我们会学习函数作用域、eval 作用域，但是没有块作用域。

3.3　JavaScript 对象

对象既可以是客观世界中存在的人、物体等实体，也可以是社会生活中的一种逻辑结构。对象由两个元素构成，一个是属性，一个是方法。属性表示有什么，用来描述对象；方法表示能够做什么，告诉外界对象有哪些功能。方法以属性为基础，对象的属性也可以是一个对象。比如一个人就是一个典型的对象，包含身高、体重，年龄等特性，又包含吃饭、走路、睡觉等动作。象棋规则也是一个对象，它包含各个棋盘属性，还有判定棋子移动是否有效的规则。

对象在 JavaScript 中无处不在，首先 JavaScript 本身具有许多内置对象，比如 Date、Math、Array 等。还包括浏览器提供的内置对象，比如 window、document、location 等。还包括各种各样的自定义对象。

另一方面，JavaScript 中也有一些简单的数据类型，包括数字、字符串、布尔值、null 值和 undefined 值，它们不是对象。

3.3.1　创建对象

对象包含一系列属性，这些属性是无序的。每个属性都有一个字符串 key 和对应的值，这个值可以是原始值、对象或函数。这种基本数据结构还有很多叫法，有些我们已经非常熟悉，比如“散列”(hash)、“散列表”(hashtable)、“字典”(dictionary)、“关联数组”(associativeArray)。

可以通过对象直接量、关键字 new 和 Object.create()函数来创建对象。

1. 对象的字面量

对象字面量提供了一种非常方便的创建对象的方法。一个对象字面量就是一个花括号括起来的键值对。属性名可以是 JavaScript 标识符，也可以是字符串直接量(包括空字符串)。属性的值可以是任意类型的 JavaScript 表达式，表达式的值(可以是原始值也可以是对象值)就是

这个属性的值。例如：

```
var e = {}
var point = {x:0,y:0}
var point2 = {x:point.x,y:point.y+1}
var obj = {a:27, "b":99};
var s = {"":88, "p":99};

var book = {
        "a":"a",
        author:{firstname:"F",secondname:"S"}
}
```

当 property 中出现空格、斜杠等特殊字符，或者使用的 property 与 JS 关键词冲突时，则必须使用引号。

2. 使用 new 运算符创建对象

new 运算符后面跟随一个对象类型的标识符(构造函数)，比如：

```
var user = new Object();
user.name = "a";
user.age = 21;
console.log(user);//输出 Object {name: "a", age: 21}
var myObject = new Object();
myObject['0'] = 'a';
myObject['1'] = 'b';
myObject['2'] = 'c';
console.log(myObject); //输出 Object {0: "a", 1: "b", 2: "c"}
var myString = new String('abc');
console.log(myString); //输出 String {0: "a", 1: "b", 2: "c", length: 3}
```

上面的例子中，我们首先调用构造函数创建了对象，然后再给对象的属性赋值。第一种方法调用了 Object 构造函数来实现，第二种方法使用 String 构造函数来实现。那是否可以自己定义构造函数以方便创建对象呢？当然可以，比如：

```
var Person = function(living, age, gender) {
    this.living = living;
    this.age = age;
    this.gender = gender;
};
var myPerson = new Person(true, 33, 'male');
console.log(myPerson);
```

上面的例子自定义了 Person()构造函数，并使用它创建了一个复杂的 Person 对象。

3. 使用 Object.create()创建对象

在讲述第三种对象创建技术之前，首先解释一下原型的概念。每一个 JavaScript 对象都和另一个对象相关联，这里的另一个对象就是原型对象。原型的意思就是每一个对象都从原型继承属性和方法。

我们学习过 String 对象具有 length 属性和 trim 方法，这些属性和方法是在原型对象中定义的。每个对象都有一个内部属性 prototype，我们通常称之为原型。原型的值可以是一个对象，也可以是 null。如果它的值是一个对象，则这个对象也一定有自己的原型，这样就形成了一条线性的链，我们称之为原型链。例如：

```
var myFunction = fnction () {}
console.log(myFunction.prototype);//输出 object
console.log(typeof myFunction.prototype);//输出 'object'
console.log(String.prototype);//输出 String {strLen: function, strToChars:
    function, isCHS: function, subCHString: function, subCHStr: function…}
console.log(typeof String.prototype); //输出 'object'
```

默认的 prototype 属性是 Object()对象，而通过 new String()创建的对象的原型就是 String.prototype。由于 prototype 属性是一个对象，因此原型链查找到最后是 Object.prototype。所以，由 new String()创建的 String 对象的属性同时继承自 String.prototype 和 Object.prototype。

当访问一个对象的某一属性的，如果这个属性在此对象中不存在，就在它的 prototype 属性所指的原型对象的属性中寻找。如果找到则返回，否则继续沿着 prototype 链一直找下去，直到 prototype 为 null 的时候停止。例如：

```
function foo(){}
foo.prototype.z = 13;
var obj = new foo();
obj.y  = 12;
obj.x = 11;
console.log(obj.x);//1
console.log(obj.y);//2
console.log(obj.z);//3
```

ECMAScript 5 中引入了一个新方法：Object.create。可以调用这个方法来创建一个新对象。新对象的原型就是调用 create 方法时传入的第一个参数，例如：

```
var a = {a: 1}; // a ---> Object.prototype ---> null
var b = Object.create(a);// b ---> a ---> Object.prototype ---> null
console.log(b.a); // 1 (继承而来)
var c = Object.create(b);
// c ---> b ---> a ---> Object.prototype ---> null
```

如果想创建一个没有原型的新对象，可以传入参数 null，新创建的对象不会继承任何东西，甚至不包括基础方法(如 toString())，比如：

```
var d = Object.create(null);// d ---> null
console.log(d.toString())//TypeError: undefined is not a function
```

如果想创建一个普通的空对象(比如通过{}或 new Object()创建的对象)，需要传入 Object.prototype，比如：

```
var o2 = Object.create(Object.prototype)
```

3.3.2　属性的查询和设置

访问对象的属性和设置对象的属性可以使用点运算符或方括号运算符，比如：

```
var obj = {x:2,y:3};
obj.x;
obj["y"];
obj["x"] = 4;
obj.y = 5;
```

这两种方式的不同之处是方括号内还可以是一个计算结果为字符串的表达式，比如：

```
var obj = {y1:1,y2:2};
var i = 1, n = 2;
for(;i<n;i++){
    console.log(obj['x'+i]);
}
```

如果读取 undefined 对象的属性，则会出现属性读写异常。

```
var obj = {x:1};
obj.y;
obj.y.z =2;
```

可以先对对象进行判断，再访问其属性，比如：

```
if( obj.y) {
    var yz = obj.y.z;
}
```

3.3.3　删除属性

delete 运算符用来删除一个属性的定义，delete 的返回值不是表示删除成功了，而是表示没有这个属性了。例如：

```
var obj = new Object();
obj.name='zs';
delete obj.name;// 输出 true
obj.name;// 输出 undefined
delete obj.name//输出 true;
```

delete 只是断开属性和宿主对象的联系，而不会在内存中删除该属性。比如：

```
var a = {};
a.b = {x:1,y:2};
var c = a.b;
delete a.b;
console.log(c.x);//输出 1
```

3.3.4　检测属性

有时我们需要对属性做一些检测，比如对象是否拥有某个属性，这个属性是对象自己的还是原型对象的，这个属性是否可以枚举等。

属性不仅包含标识它们可枚举的特性，还包括它们是否可写、可配置的特性。可枚举特性控制着当使用 for/in 遍历对象的属性时是否可以遍历该属性，可配置特性控制着其他特性(包括属性是否可以删除)的修改。比如：

```
var cat = new Object();
cat.legs = 4;
cat.name = "tom";
'legs' in cat;//true
'abc' in cat;//false
"toString" in cat;//true
cat.hasOwnProperty('legs');//true
cat.hasOwnProperty('toString');//false
cat.propertyIsEnumerable('legs');//true
cat.propertyIsEnumerable('toString');//false
```

如果想查看属性是否可枚举、可写、可配置，可以使用 Object.getOwnPropertyDescriptor()得到对属性的描述信息。属性的可枚举、可写、可配置默认情况下都是 false。它的语法为：

```
Object.getOwnPropertyDescriptor(obj, prop)
```

如果想设置属性的可枚举、可写、可配置，可以使用 defineProperty，它的语法是：

```
Object.defineProperty(obj, prop, descriptor)
```

第一个参数是需要定义属性的对象；第二个参数是需要被定义或修改的属性名；第三个参数是一个对象，描述需要被定义或修改的属性。

```
var a = {x:1};
Object.getOwnPropertyDescriptor(a,'x');
Object.defineProperty(a,'y',{value:2,writable:false,enumerable:false,
    configurable:true});
a.y = 3;
a.y;//输出 2
Object.defineProperty(a,'y',{value:3,writable:false});
a.y; //输出 3
```

3.3.5　JavaScript 数组

数组是值按一定顺序排列的集合，每个值叫做一个元素，用每个元素的数字编号表示元素在数组中的位置，也就是索引。第一个元素的索引是 0，最大可能的索引为 4 294 967 294。JavaScript 中的数组是弱类型的，数组中可以含有不同类型的元素。数组元素甚至可以是对象或其他数组。

既可以通过字面量创建数组，也可以通过构造函数创建数组。比如：

```
var emptty = [];
var a = new Array();
var a=new Array(1,null,undefined,true,'testing',{x:1},[2,3,4]}
var a=new Array(10);
```

请注意最后一行代码是创建了一个数组并指定长度，并不是创建了只包含元素 10 的数组。

我们可以使用索引对数组的元素进行读和写，使用方括号运算符时，方括号中是一个返回非负整数值的任意表达式。同时数组的长度是动态改变的，也就是 length 属性会根据数组的元素个数进行自动更新。比如：

```
var arr = [1,2,3,4,5];
arr[1];//2
arr.length;//5
arr[5] = 6;
arr.length;//6
delete arr[0];
arr[0];//undefined
```

数组有许多常用的属性和方法，具体见表 3-5。

表 3-5　数组的常用属性和方法

属性或方法	说　　明
length 属性	用来获取数组的长度。数组的位置同样是从 0 开始的
toString 方法	将数组转换为字符串
contact 方法	用于连接两个或多个字符串
push 方法	在数组的末端添加一个或多个元素，并返回添加后的数组的长度
pop 方法	用于删除数组的最后一个元素，并返回该元素
join 方法	以参数作为分隔符，将所有数组成员组成一个字符串返回。如果不提供参数，默认用逗号分隔
shift 方法	用于删除数组的第一个元素，并返回该元素
unshift 方法	在数组的第一个位置添加元素，并返回添加新元素后的数组长度
reverse 方法	用于颠倒数组中元素的顺序，使用这个方法以后，返回改变后的原数组
slice 方法	返回指定位置的数组成员组成的新数组，原数组不变。它的第一个参数为起始位置(从 0 开始)，第二个参数为终止位置(但该位置的元素本身不包括在内)
splice 方法	用于删除元素，并可以在被删除的位置添加新的数组元素。它的返回值是被删除的元素。需要特别注意的是，该方法会改变原数组。它的第一个参数是删除的起始位置，第二个参数是被删除的元素个数
sort 方法	该方法对数组元素进行排序，默认是按照字典顺序排序。排序后，原数组将被改变

下面的例子使用了数组的属性和方法：

```
var cities = new Array("shandong","henan","gz");
console.log(cities.length);//3
```

```
console.log(cities.toString());//shandong,henan,gz
cities.length  -= 1;
console.log(cities);// shandong,henan

var arr = [];
arr[0] = 1;
arr[1] = 2;
arr.push(3);
arr;//[1,2,3]

arr[arr.length] = 4;
arr;// [1,2,3,4]

arr.unshift(0);
arr//[0,1,2,3,4]
delete arr[2];
arr;//[0,1,undefined,3,4]
arr.length;//5
arr.pop();//4 returned
arr;//[0,1,undefined,3]
arr.shift();
arr;//[1, undefined,3]

var a = ['a', 'b', 'c'];
a.reverse() // ["c", "b", "a"]
a // ["c", "b", "a"]
a.sort();//["a", "b", "c"]
a;// ["a", "b", "c"]
```

3.3.6　JavaScript 内置对象

JavaScript 实际上预定义了若干原生构造函数，共包括 9 个原生对象的构造函数，包括 Number()、String()、Boolean()、Object()、Array()、Function()、Date()、Math()、RegExp()、Error()。由于在前文中已经介绍过一些内置对象，下面重点介绍 Date()、Math()、RegExp()、Error()。

1. Date 对象

Date 对象是 JavaScript 提供的日期和时间的操作接口。通过 Date 对象既可以得到本地时间，也可以得到 UTC 时间(国际统一时间)，还可以单独处理年、月、日等信息。例如：

```
var t1 = new Date();
var t2 = new Date("Jun 7,2013 11:23:20");
var t3 = new Date(1362790014000);
console.log(t3);// 输出 Sat Mar 09 2013 08:46:54 GMT+0800 (中国标准时间)
console.log(t3.toUTCString());//输出 Sat, 09 Mar 2013 00:46:54 GMT
console.log(t3.toLocaleString());// 输出 2013/3/9 上午 8:46:54
```

2. Math 对象

在进行数学计算的时候，可以使用 Math 对象，它提供一系列数学常数和数学方法。该

对象不是构造函数，所以不能生成实例，所有的属性和方法都必须在 Math 上调用。比如：

```
new Math()// TypeError: object is not a function
Math.E //算术常量 e，即自然对数的底数 //2.718281828459045
Math.LN2//返回 2 的自然对数 0.6931471805599453
Math.LN10//返回 10 的自然对数 2.302585092994046
Math.LOG2E//返回以 2 为底的 e 的对数 1.4426950408889634
Math.LOG10E//返回以 10 为底的 e 的对数 0.4342944819032518
Math.PI//返回圆周率 3.141592653589793
Math.SQRT2 //返回 2 的平方根  1.4142135623730951
```

其中常用的方法见表 3-6：

表 3-6　Math 对象的常见方法

方　法	说　明
round	四舍五入
abs	返回参数值的绝对值
max	返回最大的参数
min	返回最小的参数
floor	小于参数值的最大整数
ceil	返回大于参数值的最小整数
random	返回 0 到 1 之间的一个伪随机数，可能等于 0，但是一定小于 1
三角函数	sin 方法返回参数的正弦，cos 方法返回参数的余弦，tan 方法返回参数的正切

3. RegExp 对象

JavaScript 中也有正则表达式，使用起来和其他语言的正则表达式类似。正则表达式(regular expression)是一种表达文本模式的方法，常常用作按照“给定模式”匹配文本的工具，比如给定一个 email 地址的模式，然后用来确定一个字符串是否为 email 地址。

新建正则表达式有两种方法。一种是使用字面量，以斜杠表示开始和结束。

语法是：

```
/pattern/attributes
```

例如：

```
var regex = /xyz/;
```

另一种是使用 RegExp 构造函数。语法是：

```
new RegExp(pattern, attributes)
```

例如：

```
var regex = new RegExp("xyz");
```

上面两种写法是等价的，都建立了一个内容为 xyz 的正则表达式对象，其中修饰符(attributes)包括 i、g、m。i 表示执行对大小写不敏感的匹配，g 表示执行全局匹配(查找所有

匹配而非在找到第一个匹配后停止)，m 表示执行多行匹配。

```
var regex = new RegExp("xyz", "i");// 等价于 var regex = /xyz/i;
```

这两种写法在运行时有一处细微的区别。采用字面量写法的正则对象在代码载入时(即编译时)生成；采用构造函数写法的正则对象在代码运行时生成。考虑到书写的便利和直观，实际应用中，基本上都采用字面量的写法。

正则表达式作为对象有如下属性：

- **ignoreCase：**返回一个布尔值，表示是否设置了 i 修饰符，该属性只读。
- **global：**返回一个布尔值，表示是否设置了 g 修饰符，该属性只读。
- **lastIndex：**返回下一次开始搜索的位置。该属性可读写，但是只在设置了 g 修饰符时有意义。
- **source：**返回正则表达式的字符串形式(不包括反斜杠)，该属性只读。
- **multiline：**返回一个布尔值，表示是否设置了 m 修饰符，该属性只读。

正则表达式对象常用的方法包括 test 方法和 exec 方法，test 方法返回布尔值，用来验证字符串是否符合某个模式。如果正则表达式带有 g 修饰符，则每一次 test 方法都从上一次结束的位置开始向后匹配。例如：

```
var r = /x/g;
var s = '_x_x';
r.lastIndex // 0
r.test(s) // true
r.lastIndex // 2
r.test(s) // true
r.lastIndex // 4
r.test(s) // false
```

相比较 test 方法，exec 方法则是返回匹配结果。例如：

```
var s = '_x_x';
var r1 = /x/;
var r2 = /y/;
r1.exec(s) // ["x"]
r2.exec(s) // null
```

上面几行代码表示，如果匹配成功，exec 方法返回一个数组，里面是匹配结果。如果匹配失败，返回 null。

当我们需要搜索一个比直接匹配需要更多条件的匹配时，比如寻找一个或多个 bc，那么这时模式将要包含特殊字符。比如，模式/ab*c/匹配了一个单独的 a 后面跟了零个或多个 b(*的意思是前面一项出现了零个或多个)，且后面跟着 c 的任何字符组合。在字符串“cbbabbbbcdebc”中，这个模式匹配了子字符串“abbbbc”。表 3-7 列出了一个在正则表达式中可以利用的特殊字符的完整列表和描述。

表 3-7　正则表达式中的特殊字符

字　符	含　义	例　子
\	对于其后的平常被当作字面量的字符，将其转义为特殊字符；对于其后的特殊字符，将其转义为字面量	/b/匹配了字符'b'。通过在 b 的前面放一个反斜杠，即用作/\b/，这个字符变成了一个特殊意义的字符，意思是匹配一个词的边界； *是一个代表着前一项 0 次或多次发生时将会被匹配的特殊字符，比如，/a*/代表会匹配 0 个或多个 a。为了匹配*号直接量，在它的前面加一个反斜杠，比如，/a*/匹配'a*'
{n}	n 是一个正整数，匹配前面一个字符刚好发生了 n 次	/a{2}/不会匹配“candy”中的'a'，但是会匹配“caandy”中所有的'a'，以及“caaandy”中的前两个'a'
{n,m}	n 和 m 都是正整数。匹配前面的字符至少 n 次，最多 m 次。如果 n 或 m 的值是 0，这个值被忽略	/a{1, 3}/并不匹配“cndy”中的任意字符，但匹配“candy”中的'a'，匹配“caandy”中的前两个'a'，也匹配“caaaaaaandy”中的前三个'a'
. (小数点)	匹配任何除了新一行开头字符的任何单个字符	
*	匹配前一个字符 0 次或多次	/bo*/会匹配“A ghost boooooed”中的'boooo'和“A bird warbled”中的'b'，但是在“A goat grunted”中将不会匹配任何东西
+	匹配前面一个字符 1 次或多次，和{1,}有相同的效果	/a+/匹配了“candy”中的'a'和“caaaaaaandy”中所有的'a'
?	匹配前面一个字符 0 次或 1 次，和{0,1}有相同的效果	/e?le?/匹配“angel”中的'el'和“angle”中的'le'
^	匹配输入的开始。如果多行标识被设置为 true，同时匹配换行后紧跟的字符	/^A/并不会匹配“an A”中的'A'，但是会匹配“An E”中的'A'
$	匹配输入的结束，如果多行标识被设置为 true，同时会匹配换行前紧跟的字符	/t$/并不会匹配“eater”中的't'，但是会匹配“eat”中的't'
\b	匹配一个词的边界。一个词的边界就是一个词不被另一个词跟随的位置或者不是另一个词汇字符前边的位置。注意，一个匹配的词的边界并不包含在匹配的内容中。换句话说，一个匹配的词的边界的内容的长度是 0	/\bm/匹配“moon”中的'm'； /oo\b/并不匹配“moon”中的'oo'，因为'oo'被一个词汇字符'n'紧跟着
\d	匹配一个数字，等价于[0-9]	/\d/或/[0-9]/匹配“B2 is the suite number.”中的'2'
\D	匹配一个非数字字符	
\w	匹配一个单字字符(字母、数字或下划线)	/\w/匹配“apple,”中的'a'、“$5.28,”中的'5' 和“3D.”中的'3'
\W	匹配一个非单字字符	/\W/或者/[^A-Za-z0-9_]/匹配“50%.”中的'%'
x|y	匹配‘x’或‘y’	/green|red/ 匹配“green apple”中的'green'和“red apple”中的'red'
\s	匹配一个空白字符，包括空格、制表符、换页符和换行符	/\W/或者/[^A-Za-z0-9_]/匹配“50%.”中的'%'

(续表)

字　符	含　义	例　子
\S	匹配一个非空白字符	/\W/或者/[^A-Za-z0-9_]/匹配“50%.”中的'%'
[xyz]	一个字符集合。匹配方括号中的任意字符。我们可以使用连接号(-)来指定一个字符范围。对于点(.)和星号(*)这样的特殊符号，在一个字符集中没有特殊的意义	[abcd]和[a-d]是一样的。它们都匹配“brisket”中的'b'，也都匹配“city”中的'c'

当这些匹配字符组合在一起的时候，会使得正则表达式不容易理解。例如[^a-z]匹配不在'a'到'z'范围内的任意字符。正则表达式还使用小括号来指定子表达式(也叫做分组)，例如(\d{1,3}\.){3}\d{1,3}是一个简单的 IP 地址匹配表达式。要理解这个表达式，请按下列顺序分析它：\d{1,3}匹配 1 到 3 位的数字，(\d{1,3}\.){3}匹配三位数字加上一个英文句号(这个整体也就是这个分组)重复 3 次，最后再加上一个一到三位的数字(\d{1,3})。

下面我们来实现登录名的客户端验证，要求登录名既可以是邮箱地址也可以是手机号，如下所示：

```
//在教材源代码中的 yanzheng.js 中
function checkMobile(mobile){
    if(mobile.length != 11 || isNaN(mobile)){
        return false;
    }
    mobile = mobile.substr(0,3);
    //号段
    var hd = new
        Array('130','131','132','133','134','135','136','137','138','139',
        '150','151','152','153','154','155','156','157','158','159','180',
        '181','182','183','184','185','186','187','188','189');
    var i = hd.length;
    while (i--) {
        if (hd[i] == mobile) {
        return true;
        }
    }
    return false;
}
//验证邮箱是否正确，//在教材源代码中的 yanzheng.js 中
function checkEmail(email){
    var reg =
        /^([a-zA-Z0-9]+[_|\_|\.]?)*[a-zA-Z0-9]+@([a-zA-Z0-9]+[_|\_|\.]?)*
        [a-zA-Z0-9]+\.[a-zA-Z]{2,3}$/;
    if(!reg.test(email)){
        return false;
    }else{
        return true;
    }
}
var loginNameCheckOk = false;
```

```
var loginName = "example@163.com";
if(checkMobile(loginName) || checkEmail(loginNameCheckOk)){
    login_name_checkOk = true;
}
```

4. Error 对象

程序在运行时经常会发生异常，会有 Error 的实例对象抛出，JavaScript 中也需要对异常进行处理。JavaScript 内置了 6 种常见的错误类型，如表 3-8 所示。

表 3-8　Error 类型

类　型	说　明
InternalError	代表 JavaScript 引擎内部错误的异常抛出的实例，例如：“递归太多”
RangeError	表示错误的原因：数值变量或参数超出其有效范围
ReferenceError	表示无效引用
SyntaxError	表示语法错误
URIError	给 encodeURI()或 decodeURl()传递的参数无效
EvalError	与 eval()函数有关

如果需要自定义 Error 对象，就需要自己创建 Error 对象，它的语法是：

```
new Error(message)
```

其中 message 表示对错误的描述信息。

3.3.7　浏览器对象

浏览器打开网页时的效果如图 3-7 所示：

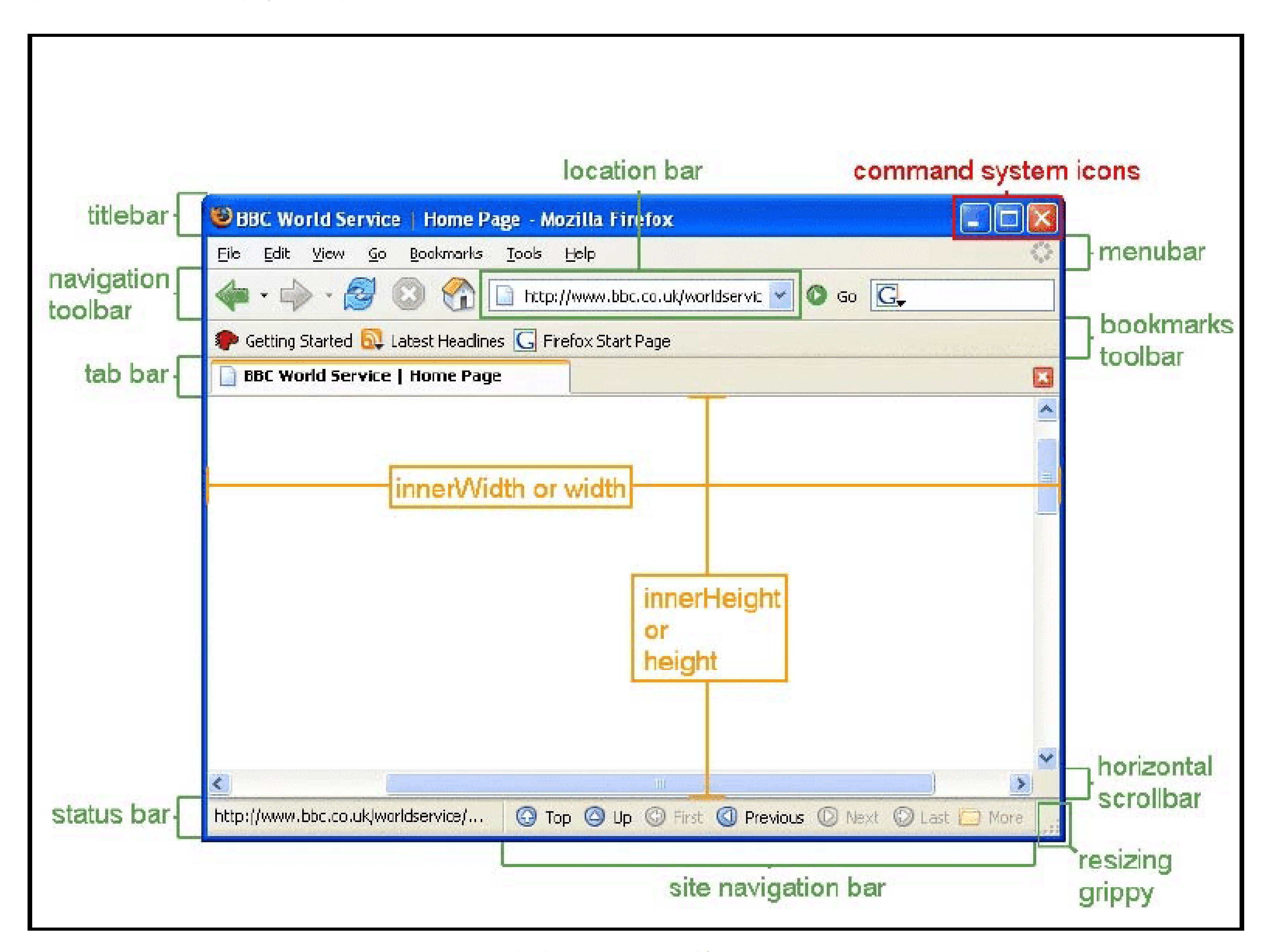


图 3-7　浏览器

大部分 JavaScript 程序是运行在浏览器环境下的，因此浏览器提供了一系列对象用于与

浏览器窗口进行交互。浏览器不仅提供了各种对象，还提供了独立于内容的描述这种对象与对象之间关系的模型(BOM)，该模型如图 3-8 所示。

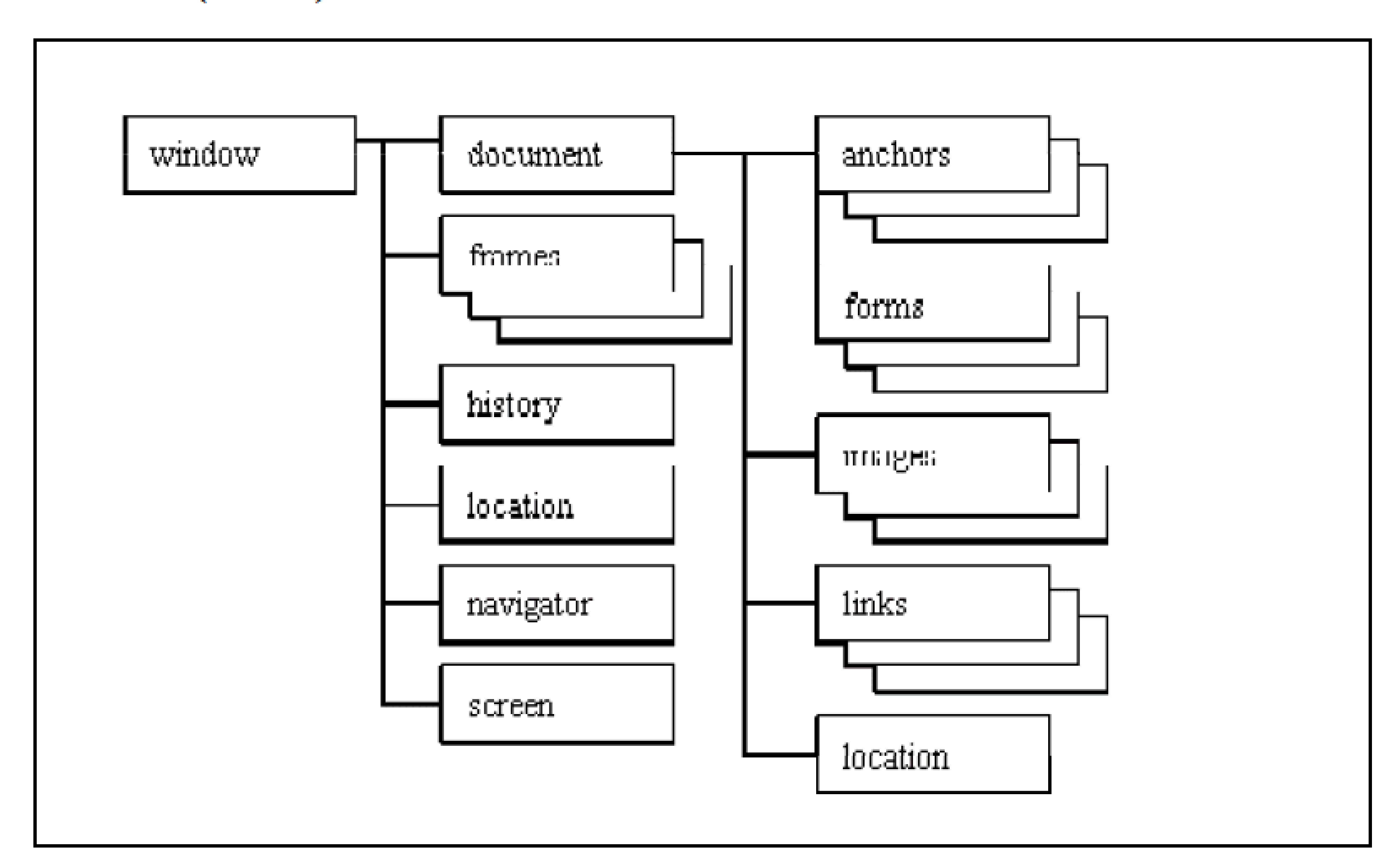


图 3-8　BOM

从图 3-8 中可以看到，window 对象是 BOM 的顶层对象。window 对象表示整个浏览器窗口，它就是 JavaScript 运行时的宿主对象，所以在浏览器窗口下执行 window 对象的属性和方法是不需要特别指明的。可以把那个窗口的属性作为全局变量来使用。例如，可以只写 document，而不必写 window.document。同样，可以把当前窗口对象的方法当成函数来使用，如只写 alert()，而不必写 window.alert()。

window 对象提供的 open 方法可以打开一个新的浏览器窗口或查找一个已命名的窗口。它的语法是：

```
window.open([url][.target][,options])
```

第一个参数是一个可选的字符串，声明了要在新窗口中显示的文档的 URL。如果省略了这个参数，或者它的值是空字符串，那么新窗口就不会显示任何内容。第二个参数也是一个可选的字符串，声明了新窗口的名称，浏览器提供了一些有用的值，比如_blank(在新窗口中打开)、_self(替换当前页面打开)。比如：

```
<html>
<head>
<script type="text/javascript">
function open_win()
{
window.open("http://www.w3school.com.cn","_blank","toolbar=yes,
    location=yes, directories=no, status=no, menubar=yes, scrollbars=yes,
    resizable=no, copyhistory=yes, width=400, height=400")
}
</script>
</head>
<body>
<form>
<input type="button" value="打开窗口" onclick="open_win()">
```

```
</form>
</body>
</html>
```

如果想关闭当前窗口，可使用 window.close()。

此外还有定时器函数 setInterval()和 setTimeout()方法，setInterval()方法可以每隔一段时间就重复执行代码，setTimeout()方法能在一段时间之后执行指定的代码。它们的语法分别为：

```
setInterval(code,millisec)
setTimeout(code,millisec)
```

在 HTML5 中，window 对象新增了两个对象：localStorage 和 sessionStorage 对象。这两个对象都能在客户端存储数据，这解决了 cookie 不适合存储大量数据的问题。

localStorage 在本地长期保存一个站点的公共变量。例如：

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
    if (localStorage.pagecount)
    {
    localStorage.pagecount=Number(localStorage.pagecount) +1;
    }
    else
    {
    localStorage.pagecount=1;
    }
    document.write("Visits: " + localStorage.pagecount + " time(s).");
</script>
<p>刷新页面会看到计数器在增长。</p>
<p>请关闭浏览器窗口，然后再试一次，计数器会继续计数。</p>
</body>
</html>
```

sessionStorage 针对一个 session 进行数据存储。当用户关闭浏览器窗口后，数据会被删除。例如：

```
<!DOCTYPE html>
<html>
<body>
<script type="text/javascript">
    if (sessionStorage.pagecount)
    {
    sessionStorage.pagecount=Number(sessionStorage.pagecount) +1;
    }
    else
    {
    sessionStorage.pagecount=1;
    }
    document.write("Visits " + sessionStorage.pagecount + " time(s) this session.");
```

```
</script>
<p>刷新页面会看到计数器在增长。</p>
<p>请关闭浏览器窗口，然后再试一次，计数器已经重置了。</p>
</body>
</html>
```

对于不同的网站，数据存储于不同的区域，并且一个网站只能访问其自身的数据。

document 对象代表 HTML 文档，每个载入浏览器的 HTML 文档都会成为 document 对象。它使我们可以从脚本中对 HTML 页面中的所有元素进行访问。例如，对于下面的 HTML 代码：

```
<html>
<head></head>
<body>
<img src = "logo.gif" name = "logo"/>
<form  method="post"  action=""  name="data">
    <input  type="email"  name="txtEmail"  />
    <input  type="submit"  value="提交"/>
</form>
</body>
</html>
```

我们可以使用 document.body 查看整个 body 标签中的内容，如果要访问 name 属性为 logo 的 img 图像，可使用 document.images["logo"]。一些常用的 document 对象的属性如表 3-9 所示。

表 3-9　document 对象的属性

属　　性	说　　明
domain	返回当前文档的域名
cookie	设置或返回与当前文档有关的所有 cookie
forms	返回对文档中所有 Form 对象的引用
lastModified	返回文档被最后修改的日期和时间
links	返回对文档中所有 Area 和 Link 对象的引用
URL	返回当前文档的 URL
images[]	返回对文档中所有 Image 对象的引用
all	提供对文档中所有 HTML 元素的访问

为了操作 HTML 文档中的各种元素，document 对象提供了一些常用的方法。首先是 document.getElementById()，它只有一个参数：我们想获得的那个元素的 id 属性，这个 id 值必须放在单引号或双引号里，它返回的是一个对象。还有与之对应的 getElementsByName()方法，它的唯一参数是想要查询的元素的 name 属性，所有 getElementsByName()方法返回的是元素的数组。此外，getElementsByTagName()方法的使用率也挺高，它也只有一个参数：元素的名字。它返回一个对象数组，每个对象分别对应着文档里具有给定标签名的一个元素。例如：

```
<html>
<head>
<script type="text/javascript">
function getValue()
```

```
{
  var x=document.getElementById("myHeader")
  alert(x.innerHTML)
}
function getElements()
{
  var x=document.getElementsByName("myInput");
  alert(x.length);
  var y=document.getElementsByTagName("input");
  alert(y.length);
}
</script>
</head>
<body>
<h1 id="myHeader" onclick="getValue()">This is a header</h1>
<p>Click on the header to alert its value</p>
<input name="myInput" type="text" size="20" /><br/>
<input name="myInput" type="text" size="20" /><br/>
<input name="myInput" type="text" size="20" /><br/>
<br/>
<input type="button" onclick="getElements()"
value="How many input elements?" />
</body>
</html>
```

3.4　JavaScript 函数

在 JavaScript 中，函数也是对象，函数拥有属性。它可以存在于一个变量、数组或对象中，并可以作为参数传递给函数并由函数返回。

3.4.1　创建函数

创建函数有三种方式：函数声明、函数表达式和 Function 构造函数。

函数声明的语法是：

```
function name([param[, param[, ... param]]]) {
   statements
}
```

name 是函数名，param 是参数的名称。一个函数最多可以有 255 个参数。statements 是函数的函数体。

函数表达式的语法是：

```
var func = function [name]([param] [, param] [..., param]) {
   statements
}
```

Function 构造函数的语法是：

```
[new] Function (arg1, arg2, ... argN, functionBody)
```

arg1, arg2, ... , argN 是构造函数的参数名。functionBody 是一个字符串，包含了组成函数的函数体的一条或多条语句。

下面的例子将展示这三种方式：

```
//函数构造函数
var addConstructor = new Function('x', 'y', 'return x + y');
// 函数语句
function addStatement(x, y) {
    return x + y;
}
// 函数表达式
var addExpression = function(x, y) {
    return x + y;
};
console.log(addConstructor(2,2), addStatement (2,2), addExpression (2,2));
//输出 '4 4 4 '
```

3.4.2　调用函数

在调用函数时，如果传递给函数的参数是值类型的，函数改变了这个参数的值，这样的改变不会影响函数外部。如果函数的参数是一个对象，函数改变了对象的值，这样的改变对函数外部是可见的，比如下面的例子：

```
var number = 2;
function square(number) {
  return number * number;
}
square(2);//4
console.log(number);//2
function myFunc(theObject) {
  theObject.make = "奔驰";
}
var mycar = {make: "宝马", model: "自动挡", year: 1998},
    x,
    y;
x = mycar.make;     // x 为"宝马"
myFunc(mycar);
y = mycar.make;     // y 为"奔驰"
```

如果重新给参数分配一个对象，并不会对函数外部有任何影响，因为这样只是改变了函数局部变量的值，而不是改变了对象的一个属性值。例如：

```
function myFunc(theObject) {
  theObject = {make: "奔驰", model: "手动挡", year: 2006};
}
var mycar = {make: "宝马", model: "自动挡", year: 1998},
```

```
    x,
    y;
x = mycar.make;     // x 为宝马
myFunc(mycar);
y = mycar.make;     // x 为宝马
```

函数实例包括的属性有 arguments(表示传递给函数的参数的类数组对象)、constructor(表示创建实例的构造函数的引用)。比如：

```
var add = function() {
    return arguments[0] + arguments[1];
};
console.log(add(4, 4)); // 输出 8
```

函数实例包括的方法有 apply 和 call 方法。

apply 方法的语法是：

```
function.apply([thisObj[,argArray]])
```

call 方法的语法是：

```
function.call([thisObj[,arg1[, arg2[, [,.argN]]]]])
```

这两个方法实现同样的功能，即调用一个对象的函数并将这个函数对象上下文从初始的上下文改变为由 thisObj 指定的新对象，区别是函数调用时，参数传递的不同，前者传递多个参数组成的数组，后者传递多个分开的参数。例如：

```
var greet = {
    runGreet: function(){
        console.log(this.name,arguments[0],arguments[1]);
    }
}
var cody = {name:'cody'};
var lisa = {name:'lisa'};
//在 cody 对象上调用 runGreet 函数
greet.runGreet.call(cody,'foo','bar'); //输出 'cody foo bar'
//在 lisa 对象上调用 runGreet 函数
greet.runGreet.apply(lisa, ['foo','bar']); //输出 'lisa foo bar'
```

在函数内部存在 this 关键字，JavaScript 中的 this 关键字有特殊的含义，它用来引用包含函数的对象，而不是函数本身(使用 new 关键字或 call 和 apply 的情况除外)。也就是说，this 的值基于调用函数的上下文。例如：

```
var  ok  =  'ok';
var  obj  =  {ok:'I am ok '};
var sayOk  =  function  ()  {
     console.log(this['ok']);
}
obj.sayOk  = sayOk;
obj.sayOk();//I am ok
sayOk(); //ok
```

3.4.3　匿名函数

函数表达式可以用来创建匿名函数，例如：

```
//function(){console.log('hi');}; // 匿名函数，但无法调用
// 创建一个函数来执行匿名函数
var sayHi = function(f){
    f(); // 调用匿名函数
}
// 将匿名函数作为参数传递
sayHi(function(){console.log('hi');}); // 输出 'hi'
```

3.4.4　作用域和闭包

在书写 JavaScript 时有一个容易忽视的问题，在函数内部使用 var 定义的变量是局部变量，而省略 var 时定义的变量则是全局变量。例如：

```
function test(){
  message="hi";
}
test();
alert(message); // "hi"
alert(window.message); "hi"
```

这里就有作用域的概念，JavaScript 中的作用域是执行代码的上下文。函数内定义的变量不能从函数之外的任何地方取得，也就是函数内部可以访问函数运行上下文的任何变量。例如：

```
var num1 = 20,   num2 = 30,   name = "Tom";
function add() {
  return num1 + num2;
}
add(); // 输出 50
function getScore () {
  var num1 = 2,
      num2 = 3;
  function add() {
    return name + " scored " + (num1 + num2);
  }
  return add();
}
getScore(); // 输出 "Tom scored 5"
```

在 JavaScript 中查找变量的值，会遵循作用域的层次结构。如果找不到该变量，就在父函数中查找，直到在全局作用域中查找。例如：

```
var x1 = 1;
var score = function() {
    var x2 = 2;
    var bar = function() {
```

```
            var x3 = 3;
            console.log(x1 + x2 + x3);
        }();//马上执行函数的写法
    };
    score();//输出 6
```

这样就形成了作用域链，而且这种“链”是由函数定义时的位置确定的。这样就形成了闭包，闭包由两部分构成：函数以及创建该函数的环境。也就是定义在闭包中的函数可以“记忆”它创建时候的环境。例如：

```
var add = function (a) {
    return function (b) {
        return a + b;
    };
};
var addFive = add(5);
alert(addFive(10));//输出 15
```

3.5　事件处理

3.5.1　浏览器事件介绍

无论是 GUI 还是 Java Swing 或.NET Framework 实现的桌面程序，程序的步骤基本上都是一样的：第一步建立用户界面，第二步等待一些事情发生，第三步做出相应的反应并转到第二步。在 HTML 网页中用户的操作产生大量的事件，比如移动或单击鼠标、键盘输入、iPhone 手势等。如果没有事件的响应能力，万维网的最大用途可能仅是显示页面。

最早的浏览器事件模型是网景事件模型，是网景浏览器中引入的一个事件模型，也是大多数网页开发者使用最广泛的事件处理模型。直到 2001 年 11 月 W3C 才真正为事件处理建立了标准模型，这个模型得到了大部分现代浏览器的支持，但是 IE 浏览器却特立独行地不支持。

window 常用事件如表 3-10 所示。

表 3-10　window 事件

事　件	说　明
onload	页面结束加载之后触发
onresize	当浏览器窗口被调整大小时触发
onunload	离开页面时触发(浏览器窗口已被关闭、使用前进或后退按钮)
onerror	在错误发生时运行的脚本
onhaschange	当文档已改变时运行的脚本
onafterprint	文档打印之后运行的脚本
onbeforeprint	文档打印之前运行的脚本
onpageshow	当窗口成为可见时运行的脚本

常用的鼠标事件如表 3-11 所示。

表 3-11　鼠标事件

事　　件	说　　明
onclick	元素上发生鼠标点击时触发
ondblclick	元素上发生鼠标双击时触发
onmousedown	当元素上按下鼠标按钮时触发
onmousemove	当鼠标指针移动到元素上时触发
onmouseout	当鼠标指针移出元素时触发
onmouseover	当鼠标指针移过元素时触发
onmouseup	当在元素上释放鼠标按钮时触发
ondrag	元素被拖动时运行的脚本
ondragend	在拖动操作末端运行的脚本
ondragenter	当元素已被拖动到有效拖放区域时运行的脚本
ondragleave	当元素离开有效拖放目标时运行的脚本
ondragover	当元素在有效拖放目标上正在被拖动时运行的脚本
ondragstart	在拖动操作开端运行的脚本
ondrop	当被拖元素正在被拖放时运行的脚本
onscroll	当元素滚动条被滚动时运行的脚本

3.5.2　处理事件的两种方法

关于事件模型的实现和讨论有许多，JavaScript 的事件模型比较独特，它完全是异步的。我们只需要简单地使用事件处理函数注册一个回调函数就行了，一旦事件触发，那么这个处理函数就会运行回调函数。这与线程方式实现的区别在于如何等待事件的发生，在线程中我们可能需要不断地检查条件是否满足。

JavaScript 事件模型分为两个阶段：捕获阶段和冒泡阶段，如图 3-9 所示。

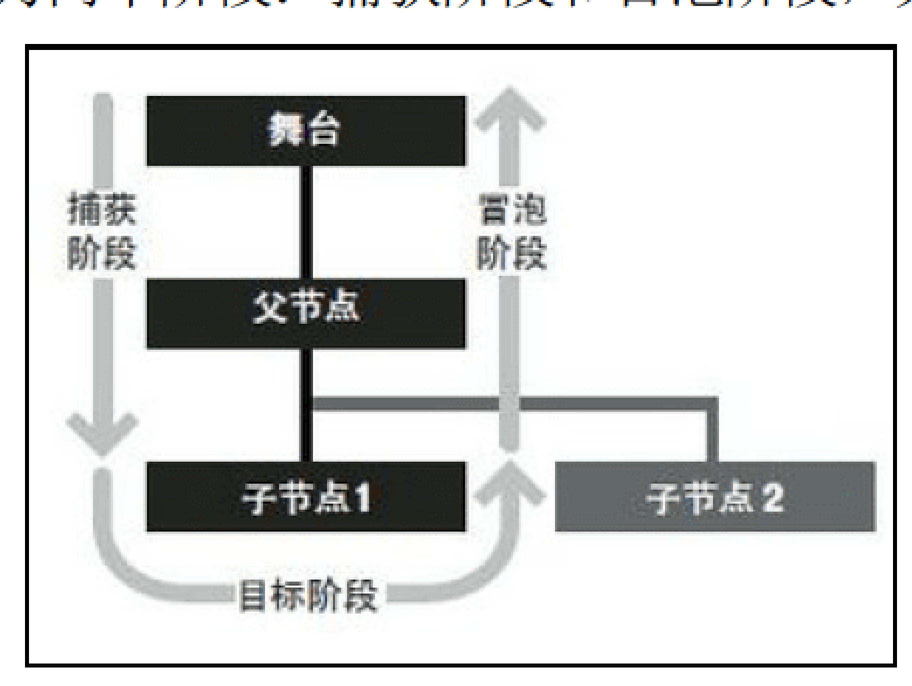


图 3-9　JavaScript 事件模型分为捕获阶段和冒泡阶段

当触发一个事件时，会产生一个事件对象，事件对象首先从舞台(document)向下传播到目标元素，然后再从目标节点上传到舞台。前一个阶段称为捕获阶段，后一个阶段称为冒泡阶段。

传统的事件绑定方式，是在一个元素上直接绑定方法。例如：

```
<body>
  <input type="button" id="bt" name="bt button" value="this is a button">
  <script>
    var bt = document.getElementById("bt");
    bt.onclick = function(e){
      alert("this is a alert");
      alert(e.currentTarget.name);
    }
  </script>
</body>
```

这是传统的事件绑定，它非常简单而且稳定，适应不同的浏览器。e 表示事件，this 指向当前元素。但是这样的绑定只会在事件冒泡中运行，事件捕获中不运行。一个元素一次只能绑定一个事件函数。

W3C 中推荐使用的事件绑定是用 addEventListener()函数，例如：

```
<body>
  <input type="button" id="bt" name="bt button" value="this is a button">
  <script>
    var bt = document.getElementById("bt");
    bt.addEventListener('click',function(e){
      alert("click once");
      alert(e.currentTarget.name);
    },false);
        bt.addEventListener('click',function(e){
      alert("click twince");
      alert(e.currentTarget.name);
    },false);

  </script>
</body>
```

如此的效果和之前的传统绑定方式是一样的，这种绑定同时支持捕获和冒泡，addEventListener()函数最后的参数表示事件处理的阶段，false 和 true 分别表示冒泡和捕获阶段。这种方式最重要的好处就是对同一元素的同一个类型事件做绑定不会覆盖，会全部生效。比如对上面代码中的 bt 元素再进行一次 click 绑定，那么两次绑定的事件处理函数都会执行，并按照代码书写顺序执行。但是 IE 浏览器不支持 addEventListener()函数，只在 IE9 以上(包括 IE9)可以使用。IE 浏览器相应地要使用 attachEvent()函数代替，IE 下事件绑定的函数 attachEvent()支持全系列的 IE。但是如果我们在 Chrome 等其他内核浏览器中使用这个函数去绑定事件，浏览器会报错。例如：

```
<body>
  <input type="button" id="bt" name="bt button" value="this is a button">
  <script>
    var name = "world";
    var bt = document.getElementById("bt");
    bt.attachEvent('onclick',function(){
```

```
        alert("hello "+ this.name);
      });
    </script>
  </body>
```

attachEvent()函数支持事件捕获的冒泡阶段，同时它不会覆盖事件的绑定。IE 中事件绑定的事件名与通用标准不同，比如通用的事件名“click”、“mouseover”在 IE 中则是“onclick”、“onmouseover”。

3.5.3　事件对象

事件对象通常在处理函数中使用，处理函数执行完之后，事件对象就自动消失了。事件对象代表事件的状态，比如键盘按键的状态、鼠标的位置、鼠标按钮的状态。在 IE 浏览器中发生一个事件时，浏览器将会自动创建一个名为“event”的事件对象，event 对象实际上又是 window 对象的一个 event 属性，因此在代码中可以通过 window.event 或 event 形式来访问该对象。而在通用模型中规定 event 对象必须作为唯一的参数传递给事件处理函数。为了兼容这两种浏览器方面的差异，可以采用下面的方法：

```
btn.onClick = function(btnEvent){
    btnEvent = btnEvent || window.event;
}
```

根据事件的不同类型，比如是鼠标事件还是键盘事件，就能访问事件的相应属性。表 3-12 给出了事件的常用属性。

表 3-12　事件的常用属性

事　件	说　明
altKey	返回当事件被触发时，“ALT”键是否被按下
shiftKey	返回当事件被触发时，“SHIFT”键是否被按下
screenX/ screenY	返回当事件被触发时，鼠标在计算机屏幕上的坐标
clientX/ clientY	返回当事件被触发时，鼠标在浏览器窗口中的坐标
ctrlKey	返回当事件被触发时，“CTRL”键是否被按下
metaKey	返回当事件被触发时，“meta”键是否被按下
relatedTarget	返回与事件的目标节点相关的节点
bubbles	返回布尔值，指示事件是否是冒泡事件类型
target	返回触发此事件的元素(事件的目标节点)
currentTarget	返回其事件监听器触发该事件的元素
cancelable	返回布尔值，指示事件是否可取消的默认动作
timeStamp	返回事件生成的日期和时间
type	返回当前 Event 对象表示的事件的名称

事件对象的方法有 preventDefault()和 stopPropagation()。

preventDefault()通知浏览器不要执行与事件关联的默认动作。例如：

```
<body>
<script type="text/javascript">
function stopDefault( e ) {
    if ( e && e.preventDefault )
       e.preventDefault();
   else
       window.event.returnValue = false;

   return false;
}
</script>
<a href="http://www.baidu.com" id="testLink">百度</a>
<script type="text/javascript">
var test = document.getElementById('testLink');
test.onclick = function(e) {
  alert('我的链接地址是: ' + this.href + ', 但是我不会跳转。');
  stopDefault(e);
}
</script>
</body>
```

stopPropagation()表示不再派发事件，也就是阻止事件向上冒泡。例如：

```
<head>
<script>
function doSomething (obj,evt) {
 alert(obj.id);
 var e=(evt)?evt:window.event;
 if (window.event) {
  e.cancelBubble=true;     // IE 下阻止冒泡
 } else {
  e.stopPropagation();     // 其他浏览器下阻止冒泡
 }
}
</script>
</head>
<body>
<div id="parent1" onclick="alert(this.id)" style="width:250px;background-
     color:yellow">
 <p>This is parent1 div.</p>
 <div id="child1" onclick="alert(this.id)" style="width:200px;background-
     color:orange">
  <p>This is child1.</p>
 </div>
 <p>This is parent1 div.</p>
</div>
<br />
<div id="parent2" onclick="alert(this.id)" style="width:250px;background-
     color:cyan;">
 <p>This is parent2 div.</p>
 <div id="child2" onclick="doSomething(this,event);" style="width:200px;background-
     color:lightblue;">
  <p>This is child2. 阻止冒泡.</p>
 </div>
```

```
 <p>This is parent2 div.</p>
</div>
</body>
```

运行结果如图 3-10 所示。

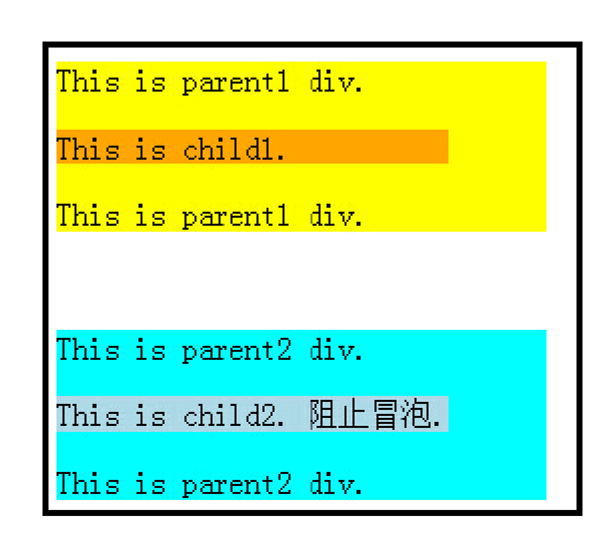


图 3-10　运行结果

图 3-10 中的第一部分代码会输出 child1 和 parent1，第二部分代码调用了 stopPropagation()函数，结果就是只输出 child2。

下面我们来实现首页的滚动条功能，这里需要使用函数 setInterval 让 div 向左循环滚动，当鼠标移动上去的时候停止滚动，这时需要 clearInterval 清除滚动效果。代码如下：

```
<html>
<head>
<meta charset="utf-8" />
<title>人人微博</title>
<link rel="stylesheet" href="styles/index.css" />
<script src="scripts/index.js"></script>
<script src="scripts/move.js"></script>
<!--[if lt IE9]>
<script
src="http://cdnjs.cloudflare.com/ajax/libs/html5shiv/3.6/html5shiv.min.js">
</script>
<![endif]-->
<script type="text/javascript">
function scrolls(obj){
   var num=(obj.scrollLeft)++;
   if(obj.scrollLeft==num){
       obj.innerHTML+=obj.innerHTML;
   }
   if(obj.scrollLeft>=obj.firstChild.offsetWidth)obj.scrollLeft=0;
}
window.onload=function(){
var myMar=setInterval("scrolls(document.getElementById('gundong'))",20);
document.getElementById('gundong').onmouseover=function(){ clearInterval(
     myMar) };
document.getElementById('gundong').onmouseout=function(){ myMar=setInterval
     ("scrolls(document.getElementById('gundong'))",20) };
}
</script>
</head>

<body>
```

```
<div id="gundong">
                <span>滚动文字消息  
                最扯分手理由有哪些  
                第三届微小说大赛  
                在天津第六天了  
                红军哥哥？</span>
              </div>
</body>
</html>
```

本章小结

本章介绍了 JavaScript 语言的基础语法，还介绍了一些概念和术语：数组、函数、表达式、原型链、闭包等。这些概念现在还不太容易理解，本章提供了相应的示例帮助我们理解。如果我们对对象的理解还不够深入，别着急。后面的章节会使用 JavaScript 的框架(jQuery)完成功能，它简化了开发者使用 JavaScript 的习惯。

本章还介绍了非常有用的浏览器事件模型，学习了 JavaScript 事件如何运作，还了解了不同浏览器绑定事件的不同方法。下一章我们将学习如何使用 JavaScript 进行图形界面开发。

本章练习

1. JavaScript 是一门什么样的语言，它有哪些特点？
2. 下面的程序输出什么？

```
1 && 3
0 && 3
```

3. 定义一个含多个参数的 Log 方法，让它代理 console.log 方法。
4. 希望获取页面中所有的 checkbox 该怎么做？

第4章　JavaScript图形

在基于 HTML5 的 Web 应用中，我们可以使用 JavaScript 图形方面的知识开发有趣的游戏。HTML5 提供的新标签允许我们做更多的互动和动画。我们会展示一个弹力球游戏，玩家可以改变球的水平和垂直位置，通过这个例子来说明如何生成动画场景。

本章内容：

- 掌握 canvas 元素的基础知识
- 使用 canvas 绘制图形和文本
- 使用 canvas 绘制图像
- 如何使用 canvas 开发 HTML5 游戏

4.1　走进 canvas 的世界

4.1.1　canvas 是什么

HTML5 元素增加了一个核心元素——canvas(画布)，它不是一个插件，而是 HTML5 的原生标记。canvas 元素不需要任何插件就能够运行 2D 甚至 3D 动画。在 canvas 元素中进行绘画，它会创建一个矩形区域，我们可以添加图片、线条、文字，当然也可以在里面绘图。默认情况下，该矩形区域的宽度为 300 像素，高度为 150 像素，用户还可以自定义设置 canvas 元素的各种特性。

4.1.2　在页面中放置 canvas 元素

HTML 页面中的最基本的 canvas 元素代码如下所示：

```
<cavas id="cvs"></canvas>
```

我们增加了一个 id 属性，一般情况下需要通过 id 属性来快速查找该 canvas 元素，因为如果没有 id 属性，我们就很难操作 canvas 元素。本章将在 canvas 画布中实现一个小游戏——弹力球游戏，界面如图 4-1 所示。

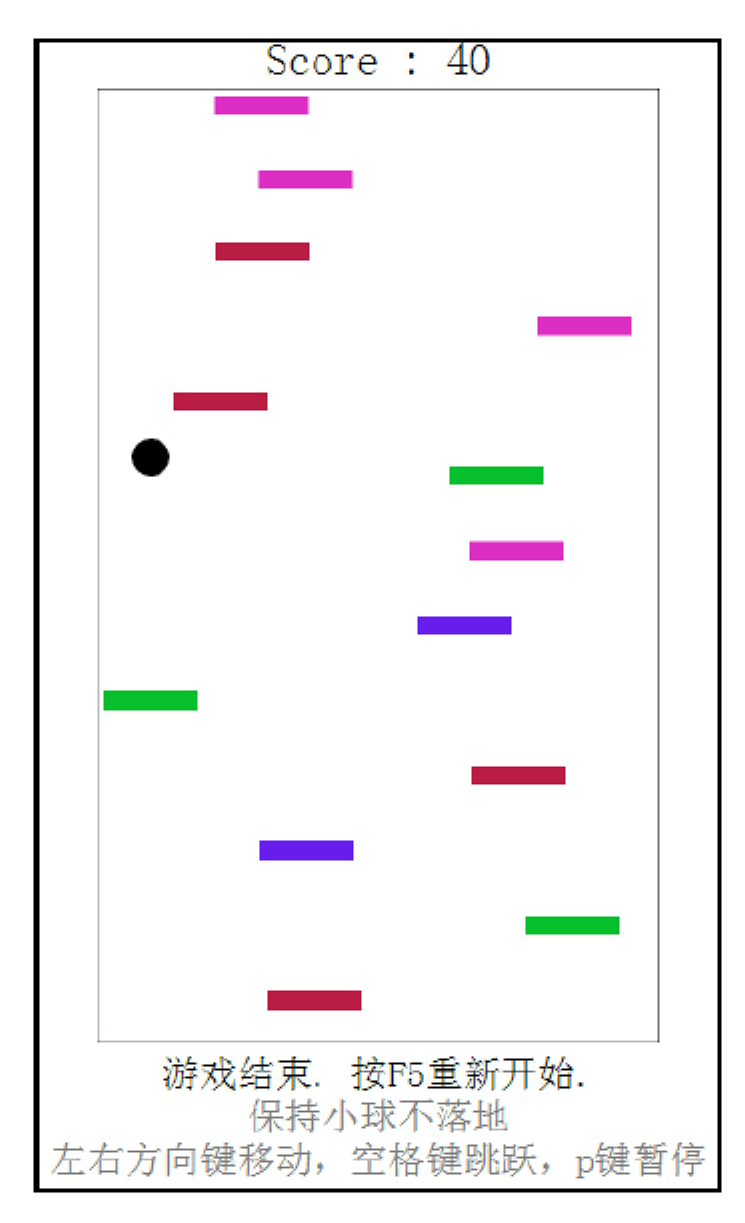


图 4-1　弹力球游戏

游戏中方块和小球不断下落，玩家的任务就是保持小球不落地，玩家需要使小球不断跳跃，当方块移出屏幕后，就会得到一定的分数。通过这个小游戏，我们来学习动画和游戏制作的原理和基础知识。游戏中都会有一个主循环，它不断地渲染画布，我们才能看到连贯流畅的画面。动态画面每秒钟展现的帧数称为帧速，它的单位是帧/秒(fps)。既可以使用 setTimeout 函数，也可以使用 setInterval 函数。代码如下：

```
//帧速
var frameSpeed = 25;
setTimeout(gameProcess,1000/frameSpeed);
function gameProcess() {
    update();
    draw();
    setTimeout(gameProcess,1000/frameSpeed);
}
//或者使用 setInterval
setInterval( function() {
    update();
    draw();
},1000/frameSpeed);
```

update 函数会执行数据的更新，draw 函数则会把新的数据绘制到画布上。这样我们就可以周期性地执行 update 函数和 draw 函数了。

首先需要确定画布的大小，在本游戏制作中，划分为两个对象：方块和小球。方块的尺寸相同，有不同的坐标和颜色，有相同的移动速度；小球的大小和颜色固定，具有向上的弹力和重力加速度。游戏中还要考虑小球与阶梯的碰撞检测、小球与 canvas 边界的碰撞检测。具体游戏设计如下：

```
var ball = {"x":10,"y":10,"dx":0,"dy":20,"acc":1,"radius":10}
var bar = {"x":10,"y":10,"color":" #000000 "}
```

```
function checkBarCollision() {}
function checkFloorCollision() {}
```

ball 对象的属性有 x、y 坐标，dx 表示 x 方向的速度，dy 表示 y 方向的速度，acc 表示小球运动中的加速度，radius 表示小球半径。

bar 对象的属性有 x、y 坐标，color 表示阶梯颜色。

checkBarCollision 函数用来检测小球与阶梯的碰撞。

checkFloorCollision 函数用来检测小球与 canvas 底部边界的碰撞。

其次我们要处理键盘事件，游戏中左右移动和小球弹起都由键盘来控制。

最后，我们需要优化游戏的设计，根据得分的高低来设置画面的帧速，这样就增加了游戏的不同难度级别。

一款游戏中需要用到一些数学物理概念、定律和公式，如果我们想踏入游戏行业，就需要了解几何知识、向量和矩阵、物理知识。

下面的代码设计了游戏中的全局变量和全局对象：

```
//canvas 宽度
var width = 300;
// canvas 高度
var height = 510;
//阶梯宽度
var barSizeWidth = 50;
//阶梯高度
var barSizeHeight = 10;
//阶梯数组
var bars = new Array();
//两个阶梯之间的距离
var step = 40;
//当前小球所在的阶梯对象
var collidedBar = null;
//屏幕滚动速度
var normalSpeed = 2;
//初始小球对象
var ball = {"x":width/2,"y":height-barSizeHeight,"dx":0,"dy":normalSpeed,
            "acc":0,"radius":10}
//玩家得分
var score = 0;
//方块颜色数组
var colors = new Array();
//重力加速度
var gravity = 0.5;
//向上的速度
var jumpVel = 9;
//小球横向移动的速度
var horizontalVel = 2;
//小球是否静止
var stop = true;
```

```
//帧速
var frameSpeed = 25;
//是否暂停
var gamePause = false;
//是否结束
var isGameOver = false;
```

下面的任务就是使用 canvas 绘制弹力游戏中的对象。

4.1.3　canvas 坐标

canvas 元素并没有创建任何可见内容，它作为一个容器，还需要 2D 渲染上下文来绘制图形，2D 渲染上下文 API(2D rendmng context API)包含绘图和图形操作所需要的全部方法和丰富功能。我们可以通过 canvas 元素访问和显示 2D 渲染上下文。可见 canvas 元素并非 canvas 动画中最强大的部分，真正的关键部分是 2D 渲染上下文，这是我们真正绘制图形的地方。

2D 渲染上下文是一种基于屏幕的标准绘图平台。与其他 2D 平台类似，它采用平面的笛卡儿坐标系统，左上角为原点(0,0)。向右移动时，x 坐标值会增加；向下移动时，y 坐标值会增加。如果我们想把图形绘制到正确的位置上，一定要理解这个坐标系。如图 4-2 所示。

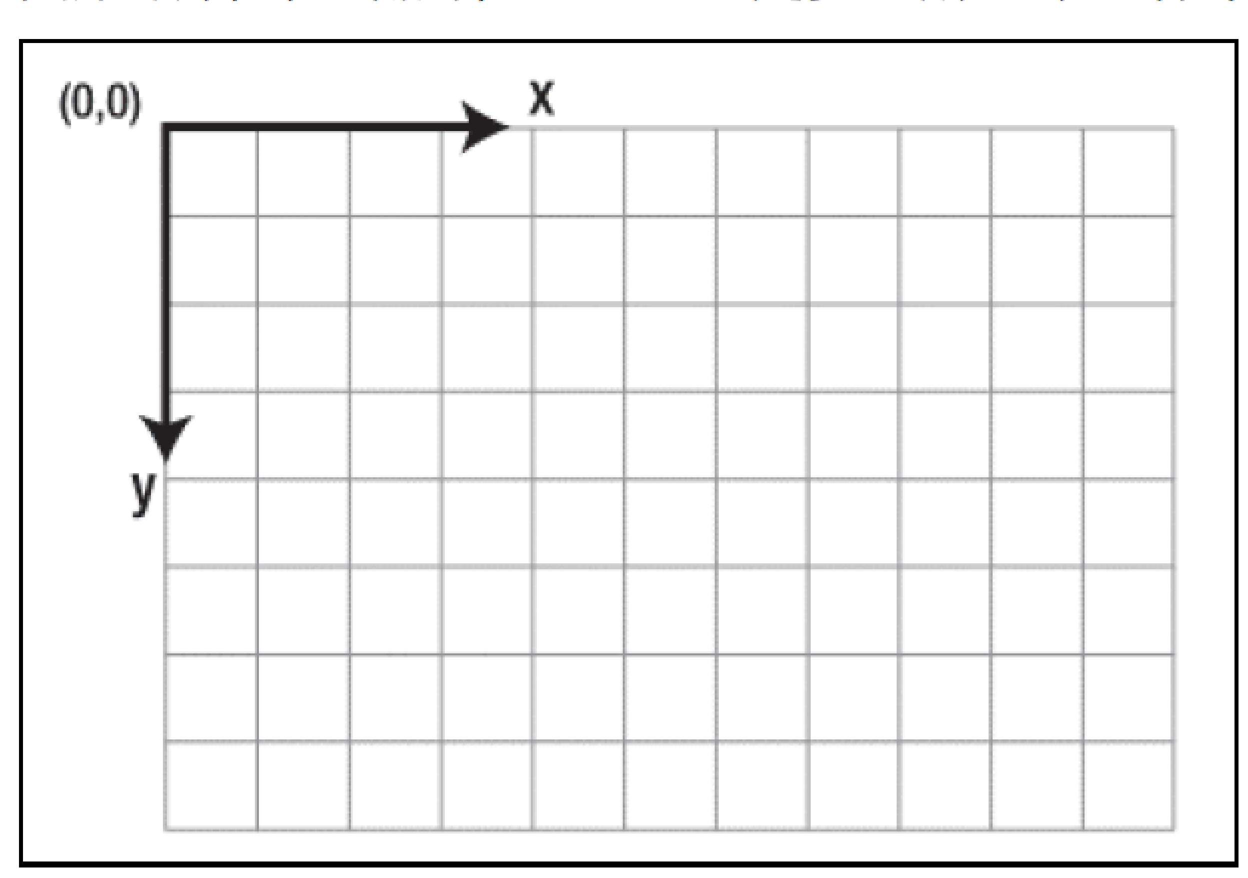


图 4-2　坐标系

canvas 元素的用途只是作为 2D 渲染上下文的包装器，它包含绘图和图形操作所需要的全部方法和丰富功能。理解这一点是很重要的，强调一下：绘图是在 2D 渲染上下文中进行的，而不是在 canvas 元素中进行的。可以通过 canvas 元素访问和显示 2D 渲染上下文。canvas 对象的 getContext 方法可以获得 2D 渲染上下文，代码如下所示：

```
var context = canvas.getContext('2d');
```

4.1.4　绘制带边框的矩形

我们取得了 canvas 上下文的对象后，就可以开始绘制图形了。只要添加下面的这行代码就可以了：

```
context.fillRect(0,0,80,100);
```

完整的代码如下所示：

```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas">your browser does not support the canvas tag </canvas>
<script type="text/javascript">
var canvas=document.getElementById('myCanvas');
var contexts.getContext('2d');
context.fillRect(0,0,80,100);
</script>
</body>
</html>
```

用 canvas 元素绘制图形有两种方式：填充(fill)和绘制边框(stroke)。填充是指填满图形内部；绘制边框是指不填满图形内部，只绘制图形的外框。这两个方法具有同样的参数，x 是指矩形起点的横坐标，y 是指矩形起点的纵坐标，width 是指矩形的宽度，height 是指矩形的高度。canvas 元素结合使用这两种方式来绘制图形。在进行图形绘制的时候，首先要设定好绘图的样式，然后调用有关的方法进行图形的绘制。默认填充黑色是不是太单调了，我们可以使用 fillStyle 属性设置填充的颜色，使用 strokeStyle 设置线条的颜色。

代码如下所示：

```
context.fillStyle = "rgb(255,0,0)";
context.fillStyle = "red";
context.fillStyle = "#FF0000";
```

上面三条语句的效果相同，都是设置填充颜色为红色。样式中的颜色值可以是“red”或“blue”这种颜色名，也可以是“#EEEEFFF”这种十六进制的颜色值。另外，也可以通过 rgb(红色值，绿色值，蓝色值)或 rgba(红色值，绿色值，蓝色值，透明度)函数来指定颜色的值。

在绘制图形的时候还可以设置图形边框的宽度，任何直线都可以通过 lineWidth 属性来指定直线的宽度，代码如下：

```
context.lineWidth = "3";
```

添加样式后的代码如下：

```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas">your browser does not support the canvas tag </canvas>
<script type="text/javascript">
var canvas=document.getElementById('myCanvas');
var context=canvas.getContext('2d');
context.fillStyle = "rgb(255,0,0)";
context.strokeStyle = "#00FF00";
context.lineWidth = "3";
context.fillRect(0,0,80,100);
context.strokeRect(0,0,80,100);
```

```
</script>
</body>
</html>
```

4.2　绘制基本图形

4.2.1　绘制线条

线条与绘制图形有一些区别。它们实际上称为路径。下面的代码将绘制一条直线。

```
context.beginPath(); // Start the path
context.moveTo(40, 40); // Set the path origin
context.lineTo(340, 40);// Set the path destination
context.closePath(); // Close the path
context.stroke(); // Outline the path
```

这段代码执行了如下步骤：

(1) 准备开始创建路径。

(2) 设置路径的原点坐标。

(3) 设置路径的终点坐标。

(4) 完成路径的绘制。

图像上下文对象的 beginPath 方法表示开始路径的创建，moveTo(x,y)表示将当前位置移动到新的参数坐标，lineTo(x,y)表示将当前位置移动到新的参数目标并且在两个坐标之间画一条直线。如果画布的子路径是打开的，closePath()通过添加一条线条连接当前点和子路径起始点来关闭它。

如果子路径已经闭合了，这个方法不做任何事情。一旦子路径闭合，就不能再为其添加更多的直线或曲线了。要继续向该路径添加，需要通过调用 moveTo()开始一条新的子路径。

4.2.2　绘制圆形

圆形的绘制要复杂许多，canvas 上下文把绘制圆形和绘制圆弧视作相同的功能，因为圆弧实际上是圆形的组成部分——首尾相连的圆弧就是圆形。在 canvas 中创建一个圆形的代码如下：

```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas">your browser does not support the canvas tag </canvas>
<script type="text/javascript">
var canvas=document.getElementById('myCanvas');
var context=canvas.getContext('2d');
context.beginPath(); // Start the path
context.arc(100, 100, 50, 0, Math.PI*2, false); // Draw a circle
```

```
context.closePath(); // Close the path
context.fill(); // Fill the path
</script>
</body>
</html>
```

rc()方法用来绘制圆弧，它的语法是：

```
arc(x, y, radius, startAngle, endAngle, counterclockwise)
```

参数的说明如表 4-1 所示。

表 4-1　arc 方法的参数说明

参　　数	说　　明
x、y	描述圆弧的圆形的圆心的坐标
radius	描述圆弧的圆形的半径
startAngle、endAngle	沿着圆指定圆弧的开始点和结束点的一个角度。这个角度用弧度来衡量
counterclockwise	圆弧沿着圆周的逆时针方向(TRUE)还是顺时针方向(FALSE)遍历

如果我们习惯使用角度，请使用下面的方法将弧度转换为角度：

```
var radians = degrees*Math.PI/180
```

其中 Math.PI 表示角度为 180 度。

4.3　绘制文本

4.3.1　文本设置

在游戏界面中，不仅有图像，还会有文本，canvas 如何处理文本呢？canvas 绘制文本其实很简单，代码如下：

```
<!DOCTYPE HTML>
<html>
<body>
<canvas id="myCanvas">your browser does not support the canvas tag </canvas>
<script type="text/javascript">
var canvas=document.getElementById('myCanvas');
var context=canvas.getContext('2d');
context.fillStyle='#00FF00';
context.font = '40px Arial';
context.fillText('Hello World',0,50);
context.font = 'italic 30px serif';
context.strokeText('Hello World',0,100);
</script>
```

```
</body>
</html>
```

有关文本的属性和方法有 fillText 和 strokeText 方法：

```
fillText(string text, int x, int y[, int maxWidth])
```

从指定坐标点位置开始绘制填充的文本文字。参数 maxWidth 是可选的，如果文本内容宽度超过该参数设置，则会自动按比例缩小字体以适应宽度。与本方法对应的样式设置属性为 fillStyle。

strokeText 方法的语法为：

```
strokeText(string text, int x, int y[, int maxWidth])
```

从指定坐标点位置开始绘制非填充的文本文字(文字内部是空心的)。参数 maxWidth 是可选的，如果文本内容宽度超过该参数设置，则会自动按比例缩小字体以适应宽度。该方法与 fillText()用法一致，不过 strokeText()绘制的文字内部是非填充(空心)的，fillText()绘制的文字是内部填充(实心)的。与本方法对应的样式设置属性为 strokeStyle。

4.3.2　文本的对齐方式

canvas 中提供了文本对齐的属性。textAlign 属性是水平方向的文字对齐，它的值包括 center、end、left、right、start。textBaseline 是竖直方向的文字对齐，它的值包括 alphabetic、bottom、hanging、ideographic、middle、top。

在水平方向上，为了看出各种对齐方式的不同，我们在文本坐标位置画一条竖线，代码如下所示：

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="300" height="200" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.
</canvas>
<script>
var c=document.getElementById("myCanvas");
var context=c.getContext("2d");
context.strokeStyle="blue";
context.moveTo(150,20);
context.lineTo(150,170);
context.stroke();
context.font="15px Arial";
// 显示不同的 textAlign 值
context.textAlign="start";
context.fillText("textAlign=start",150,60);
context.textAlign="end";
context.fillText("textAlign=end",150,80);
context.textAlign="left";
context.fillText("textAlign=left",150,100);
```

```
context.textAlign="center";
context.fillText("textAlign=center",150,120);
context.textAlign="right";
context.fillText("textAlign=right",150,140);
</script>
</body>
</html>
```

运行效果如图 4-3 所示。

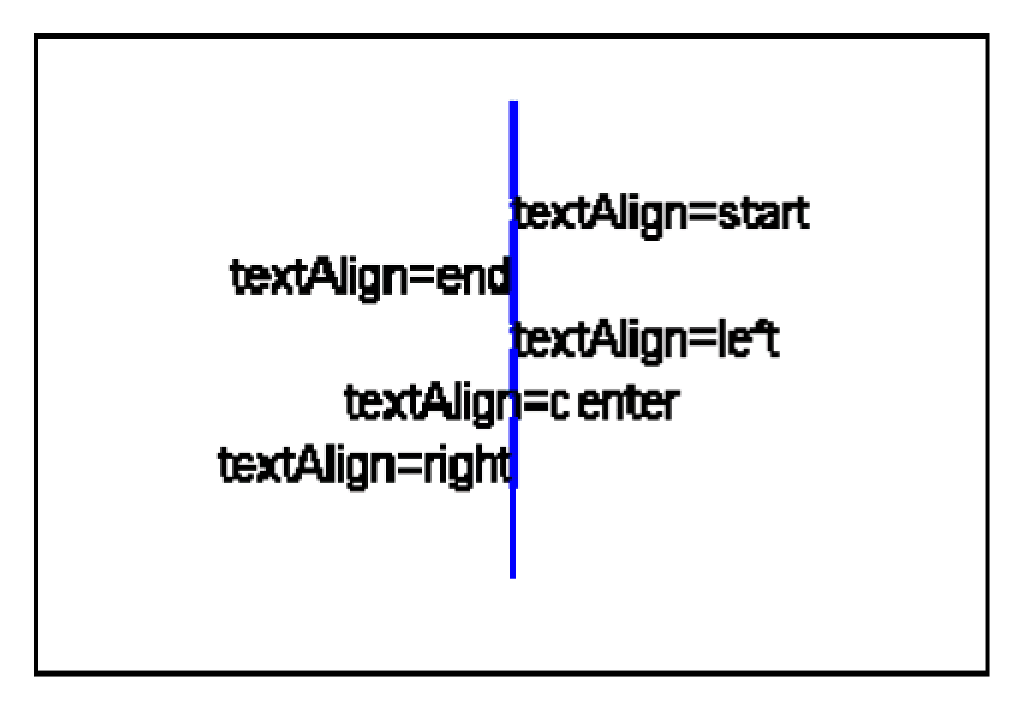


图 4-3　textAlign 的运行效果

下面我们来绘制弹力球游戏界面，首先要根据长度和宽度绘制一个画布边界，并封装为一个函数，代码如下所示：

```
//绘制画布边界
function drawRect() {
    context.beginPath();
    context.strokeRect(0,0,width,height);
    context.closePath();
    context.fill();
}
```

还需要绘制许多方块和一个小球，代码如下所示：

```
function drawBars() {
    for(var i=0;i<bars.length;i++) {
        drawBar(bars[i].x,bars[i].y,bars[i].color);
    }
}
//绘制方块
function drawBar(x,y,color) {
    context.fillStyle = color;
    context.beginPath();
    context.fillRect(x,y,barSizeWidth,barSizeHeight);
    context.closePath();
    context.fill();
}
//绘制小球位置
function drawBall(x,y,r) {
    context.fillStyle = "#000000";
```

```
        context.beginPath();
        context.arc(x,y,r,0,2*Math.PI,true);
        context.closePath();
        context.fill();
    }
```

由于画布不断重绘，重绘之前需要擦除画布，这里使用了 clearRect 方法。clearRect 方法可以清除特定区域的图形，代码如下：

```
    //清空画布
    function clear() {
        context.clearRect(0,0,width,height);
    }
```

在绘制图形之前，也就是页面加载完成后需要做初始化工作，初始方块对象和小球对象等。代码如下：

```
    window.addEventListener("load",init,true);
    //当整个页面加载完毕后此函数被首先调用
        function init() {
            context = document.getElementById("canvas").getContext("2d");
            drawRect();

            //添加颜色到全局数组，以便能够着色方块
            colors.push("#000000");
            colors.push("#BA1E45");
            colors.push("#DE2FC4");
            colors.push("#6A1FED");
            colors.push("#0ABF2E");
            colors.push("#ABB31B");
            colors.push("#C95B12");
            //初始化方块
            initBars();
           //初始化小球
            initBall();
            //循环运行下面的函数
            //setTimeout(gameProcess,1000/frameSpeed);
            timerFunction();
            scoreDiv = document.getElementById("score");
            //绑定键盘事件
            window.onkeydown = keydown;
            window.onkeyup = keyup;
        }
        //游戏开始时初始化小球
        function initBall() {
            //将小球置于画布中间的方块上
            var mid_indx = Math.floor(bars.length/2);
            ball.x = bars[mid_indx].x+barSizeWidth/2;
            ball.y = bars[mid_indx].y-ball.radius;
            collidedBar = bars[mid_indx];
        }
```

```
    //游戏开始时初始化方块
    function initBars() {
        //从上往下添加方块
        var y_bar = height-barSizeHeight;
        //计算需要的方块个数
        var noOfBars = y_bar/step;
        for(var v=0;v<noOfBars;v++) {
            //随机设置方块的 x 坐标
            var j = (width-barSizeWidth)*Math.random();
            //随机指定颜色
            bars.push({"x":Math.floor(j),"y":y_bar,"color":colors[Math.
                floor(6*Math.random())]});
            y_bar-=step;
        }
    }

/****************************Event Functions*****************************/
    function keyup(e) {
        if(e.keyCode==37 || e.keyCode==39) {
            ball.dx = 0;
        }
    }
    function keydown(e) {
        //按下空格键
        if(e.keyCode==32) {
            if(stop) {
                //检测小球是否在方块上
                if(ball.x>collidedBar.x-ball.radius&&ball.x<collidedBar.
                    x+barSizeWidth+ball.radius)
                {
                    //设置向上的速度
                    ball.dy = -jumpVel;
                    //设置重力加速度
                    ball.acc = gravity;
                    stop = false;
                }
            }
        }
        //左箭头键
        else if(e.keyCode==37) {
            if(ball.x-ball.radius!=0) ball.dx = -horizontalVel;
        }
        //右箭头键
        else if(e.keyCode==39) {
            if(ball.x+ball.radius-width!=0) ball.dx = horizontalVel;
        }
        //键盘上的'P'键
        else if(e.keyCode==80) {
            gamePause = !gamePause;
            timerFunction();
        }
```

```
    }
    function timerFunction() {
        setTimeout(gameProcess,1000/frameSpeed);
    }
```

由于 setTimeout(gameProcess,1000/frameSpeed)在多个地方使用到，上面的代码中对它进行了封装，封装为 timerFunction 函数。上面的代码首先得到 canvas 上下文，然后调用 initBars 初始化方块对象，调用 initBall 初始化小球对象，然后调用 timerFunction 函数，最后进行事件绑定，当按下空格键时执行下面的代码：

```
if(stop) {
//检测小球是否在方块上
if(ball.x>collidedBar.x-ball.radius&&ball.x<collidedBar.x+barSizeWidth+
    ball.radius)
        {
            //设置向上的速度
            ball.dy = -jumpVel;
            //设置重力加速度
            ball.acc = gravity;
            stop = false;
        }
}
```

首先通过全局变量 stop 判断小球是否处于静止状态，因为只有静止状态的小球才能响应空格键，又进一步验证小球与当前碰撞方块的位置正确后，设置小球的向上速度和重力加速度，并设置小球静止状态为 false。

下面我们来完成游戏主循环函数，代码如下：

```
//游戏的主循环函数
    function gameProcess() {
        //首先清空 canvas
        clear();
        //重画边界
        drawRect();
        //移动方块
        moveBars();
        //移动小球
        ball = moveBall(ball);
        //检查 ball 和 bar 是否碰撞
        var res = checkBarCollision();
        //如果发生碰撞
        if(res.bool) {
            //小球跟随方块移动
            ball.dy = normalSpeed;
            ball.acc = 0;
            //记录碰撞的方块，置小球停止状态为 true
            collidedBar = res.bar;
            stop = true;
        }
```

```
        //重绘小球
        drawBall(ball.x,ball.y,ball.radius);
        //重绘所有方块
        drawBars();
        //检测画布底部的碰撞
        if(checkFloorCollision()) {
            ball.dy = 0;
            ball.dx = 0;
            ball.acc = 0;
            stop = true;
        }
        //根据得到的分数设置难度
        makeDifficult();
        //如果游戏没有暂停，也没有结束。继续运行程序
        if(!gamePause&&!isGameOver)
            timerFunction();
    }
    //根据得到的分数设置难度
    function makeDifficult() {
        if(score>500&&score<1000) {
            frameSpeed = 30;
        }
        else if(score>1000&&score<1500) {
            frameSpeed = 35;
        }
        else if(score>1500&&score<2000) {
            frameSpeed = 40;
        }
        else if(score>2000&&score<3000) {
            frameSpeed = 45;
        }
        else if(score>3000&&score<4000) {
            frameSpeed = 50;
        }
        else if(score>4000&&score<5000) {
            frameSpeed = 60;
        }
    }
    //游戏结束
    function gameOver() {
        isGameOver = true;
        document.getElementById("status").innerHTML="游戏结束,按F5重新开始.";
    }
    function checkFloorCollision() {
        //检测小球与方块之间的碰撞
        if(height-(ball.y+ball.radius)<01) {
            ball.dy = 0;ball.dx=0;ball.acc=0;
            gameOver();
        }
        var right_wall = ball.x+ball.radius-width;
        var left_wall = ball.x-ball.radius;
```

```
        var bol1 = right_wall<3&&right_wall>-3;
        var bol2 = left_wall<3&&left_wall>-3;
        if(bol1||bol2) {
            ball.dx = 0;
        }
        if(ball.x>collidedBar.x||ball.x<collidedBar.x+barSizeWidth) {
            ball.acc = gravity;
        }
        return false;
    }
```

上面的代码首先清空画布并重绘边框，然后调用 moveBars 移动方块，调用 moveBall 移动小球，在移动方块的时候还需要改变玩家得分，当一个方块移出画布的底部，就增加 20 分。代码如下：

```
//移动小球
function moveBall(inpBall) {
     inpBall.x += Math.floor(inpBall.dx);
    //增加下落速度
    inpBall.dy += inpBall.acc;
    inpBall.y += Math.floor(inpBall.dy);
    return inpBall;
}
//移动方块
function moveBars() {
    //检测方块是否从 canvas 移出
    if(bars.length>0&&bars[0].y+normalSpeed==height) {
        //移除最底端的方块
        bars.shift();
        //对每个移除的方块增加 20 分
        score+=20;
        scoreDiv.innerHTML = "Score : "+score;

        //增加一个新的方块
        var j = (width-barSizeWidth)*Math.random();
        bars.push({"x":j,"y":-barSizeHeight,"color":colors[Math.floor
           (6*Math.random())]});
        }
    //增加方块的 y 坐标
    for(var i=0;i<bars.length;i++) {
        bars[i].y += normalSpeed;
    }
}
```

上面的代码通过 checkBarCollision 检测小球是否与方块相撞，checkBarCollision 返回的是一个对象，包括的属性 bool 表示是否相撞，属性 bar 表示相撞的方块对象。代码如下：

```
function checkBarCollision() {
        var response = {"bar":null,"bool":false};
        var temp_y = ball.y+ball.radius;
```

```
        for(var i = 0; i<bars.length;i++) {
            var mod = Math.floor(bars[i].y-temp_y);
            var b = mod>-5 && mod<5;
            var b1 = mod>-barSizeHeight && mod<barSizeHeight;
            //小球向下运动
            if(b1&&(ball.x>=bars[i].x && ball.x<=bars[i].x+barSizeWidth) &&
                 ball.dy>0)
            {
                ball.y = bars[i].y-ball.radius;
                response.bool  = true;
                response.bar  = bars[i];
            }
            //小球向上运动

            if(b&&(ball.x>=bars[i].x && ball.x<=bars[i].x+barSizeWidth) &&
            ball.dy<=0)
            {
                ball.y = bars[i].y-ball.radius;
                response.bool  = true;
                response.bar  = bars[i];
            }
        }
        return response;
    }
```

下面绘制下一帧界面，分别调用 drawBall 和 drawBars 绘制小球和方块。最后调用 makeDifficult 设置游戏的难度。

至此，弹力球游戏就制作完成了，并将游戏的所有代码保存为 game.html，在人人微博网站中就可以通过 game.html 访问弹力球游戏了。

4.4　使用图像

4.4.1　绘制图像

canvas 还能够加载图像到画布中，这与绘制图形一样简单。加载图像时，需要使用 drawImage 方法，该方法的语法如下所示。

```
context.drawImage(img,x,y);
```

方法只使用了三个参数，第一个参数可以是 img 元素、video 元素或者 JavaScript 中的 image 对象，使用该参数代表的实际对象来装载图像文件。x 与 y 为绘制时该图像在画布中的起始坐标。例如：

```
<!DOCTYPE HTML>
<html>
```

```
<head>
<meta charset="utf-8">
<script type="text/javascript">
window.onload = function ()
{
   var c=document.getElementById("myCanvas");
    var ctx=c.getContext("2d");
    var image = new Image();
    image.src = "tul.jpg";
    image.onload = function ()
    {
        ctx.drawImage(image,10,10);
    }
}
</script>
</head>
<body>
<p>画布：</p>
<canvas id="myCanvas" width="500" height="300" style="border:1px solid
    #d3d3d3;background:#ffffff;">
Your browser does not support the HTML5 canvas tag.
</canvas>
</body>
</html>
```

我们把 drawImage 函数放在 onload 函数里面，如果图像文件是一个来源于网络上的比较大的图像文件，必须要等待图像全部装载完毕才能绘制该图像，如果直接调用 drawImage 函数则不能正确显示该图像。

4.4.2　放大缩小图像

有时希望调整图像的大小，需要指定绘制图形的宽度和高度，方法如下所示：

```
context.drawImage(image, x, y, width, height);
```

让我们将前一个例子的图像缩小一半来显示，相应的 drawImage 方法修改为以下形式：

```
ctx.drawImage(image, 0, 0, image.width/2, image.height/2);
```

缩小后的效果跟原来的效果如图 4-4 所示。

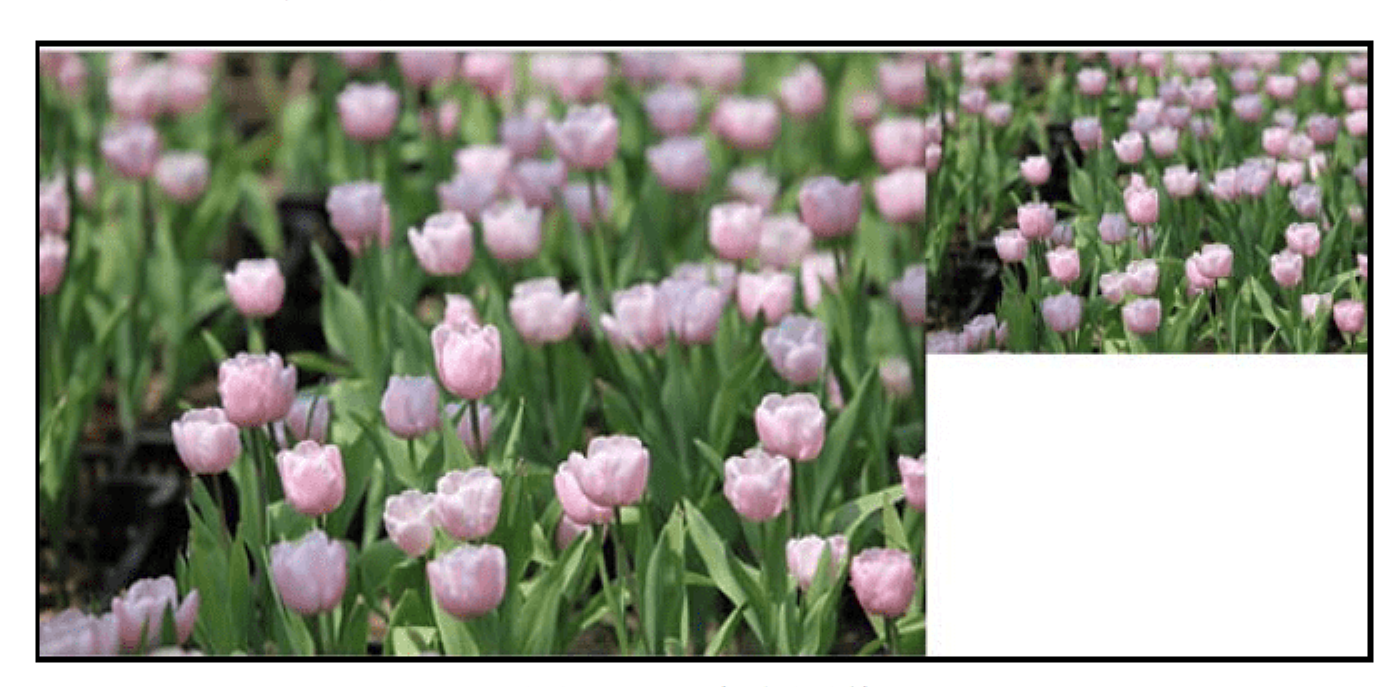


图 4-4　缩小图像

4.4.3　平铺图像

游戏制作中背景通常需要平铺，平铺技术就是按一定比例将画布铺满。可以通过 canvas 上下文的 createPattern 方法实现，语法如下：

```
context.createPattern(image,type)
```

该方法使用两个参数：image 参数为要平铺的图像；type 参数为平铺类型，它的值必须是下面的字符串之一：no-repeat(不平铺)、repeat-x(横方向平铺)、repeat-y(纵方向平铺)、repeat(全方向平铺)。例如：

```
<!DOCTYPE HTML>
<html>
<head>
   <title>平铺模式</title>
   <meta charset="utf-8">
</head>
<body>
   <canvas id="canvas" width="620" height="620" style="background-color:
       rgb(222, 222, 222)">
      Your browser does not support the HTML5 canvas tag.
   </canvas>
   <br />
   <button type="button" onclick="drawIt();">Demo</button>
   <button type="button" onclick="clearIt();">清除画布</button>
   <script type="text/javascript">
      var ctx = document.getElementById('canvas').getContext('2d');
      function drawIt() {
         clearIt();
         var img = new Image();
         img.onload = function () {
            var pattern = ctx.createPattern(img, "no-repeat");
            ctx.fillStyle = pattern;
            ctx.strokeStyle = "blue";
            ctx.beginPath();
            ctx.rect(0, 0, 300, 300);
            ctx.fill();
            ctx.stroke();

            pattern = ctx.createPattern(img, "repeat-x");
            ctx.fillStyle = pattern;
            ctx.strokeStyle = "blue";
            ctx.beginPath();
            ctx.rect(300, 0, 300, 300);
            ctx.fill();
            ctx.stroke();

            pattern = ctx.createPattern(img, "repeat-y");
            ctx.fillStyle = pattern;
            ctx.strokeStyle = "blue";
```

```
                ctx.beginPath();
                ctx.rect(0, 300, 300, 300);
                ctx.fill();
                ctx.stroke();

                pattern = ctx.createPattern(img, "repeat");
                ctx.fillStyle = pattern;
                ctx.strokeStyle = "blue";
                ctx.beginPath();
                ctx.rect(300, 300, 300, 300);
                ctx.fill();
                ctx.stroke();
            }
            img.src = "xrk.gif";
        }
        function clearIt() {
            ctx.clearRect(0, 0, 620, 620);
        }
    </script>
</body>
</html>
```

运行效果如图 4-5 所示。

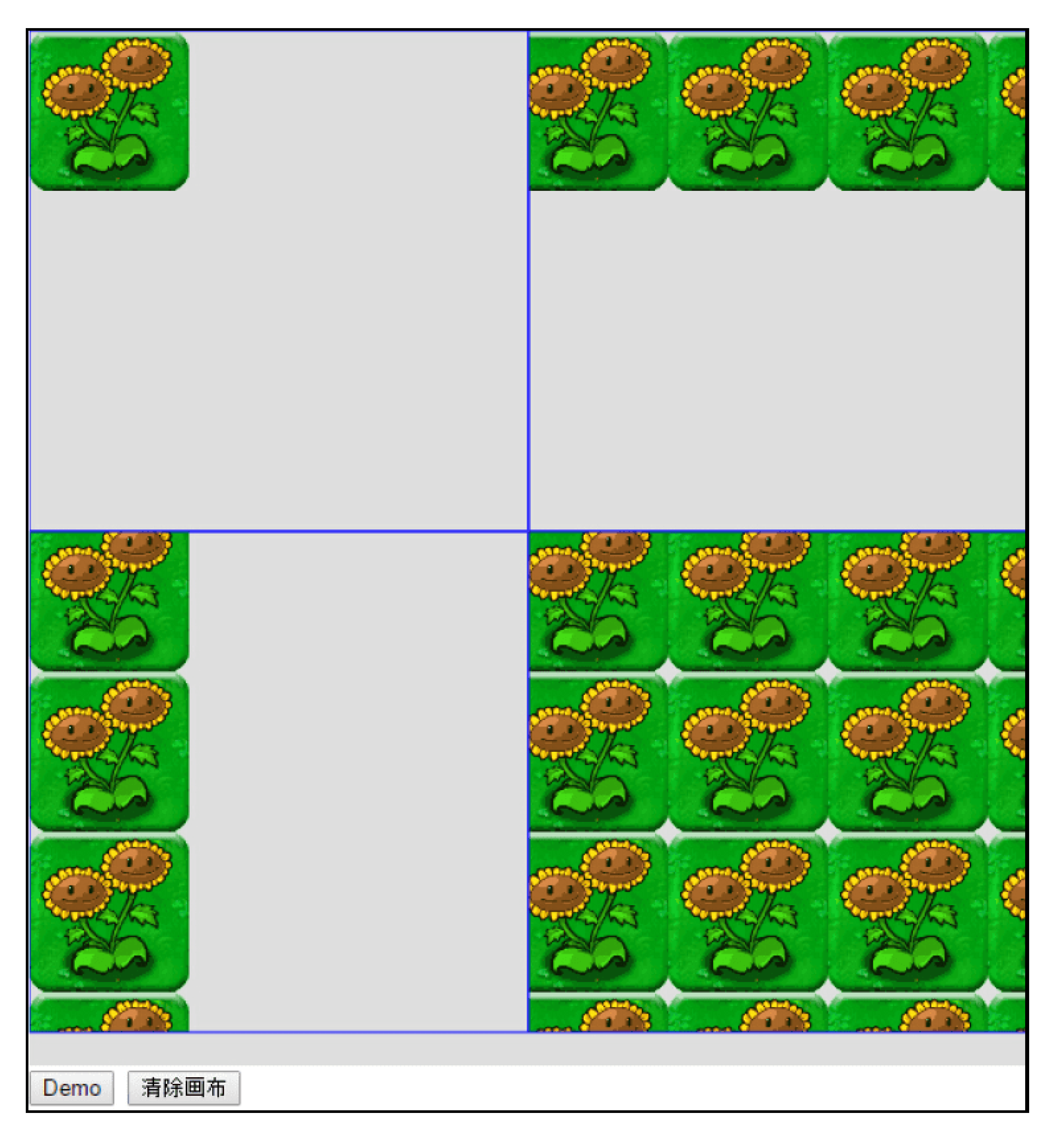


图 4-5　平铺图像的运行效果

4.4.4　裁剪图像

在绘制图像时，有时希望保留部分矩形区域，这就需要将矩形以外的全部内容删除。裁剪的目的是将图像剪切为较小尺寸，drawImage 方法还可以实现裁剪功能。语法如下所示：

```
context.drawImage(image, sx, sy, sw, sh, dx, dy, dw, dh)
```

该方法使用 9 个参数：image 仍然代表被复制的图像文件；sx 与 sy 分别表示源图像的被复制区域在画布中的起始横坐标与起始纵坐标；sw 与 sh 分别表示裁剪源图像的宽度和高度；dx 与 dy 分别表示绘制目标图像的起点坐标；dw 与 dh 表示复制后的目标图像的宽度和高度。该方法可以只复制图像的局部，只要将 sx 与 sy 设置为局部区域的起始点坐标，将 sw 与 sh 设置为局部区域的宽度和高度就可以了，如图 4-6 所示。

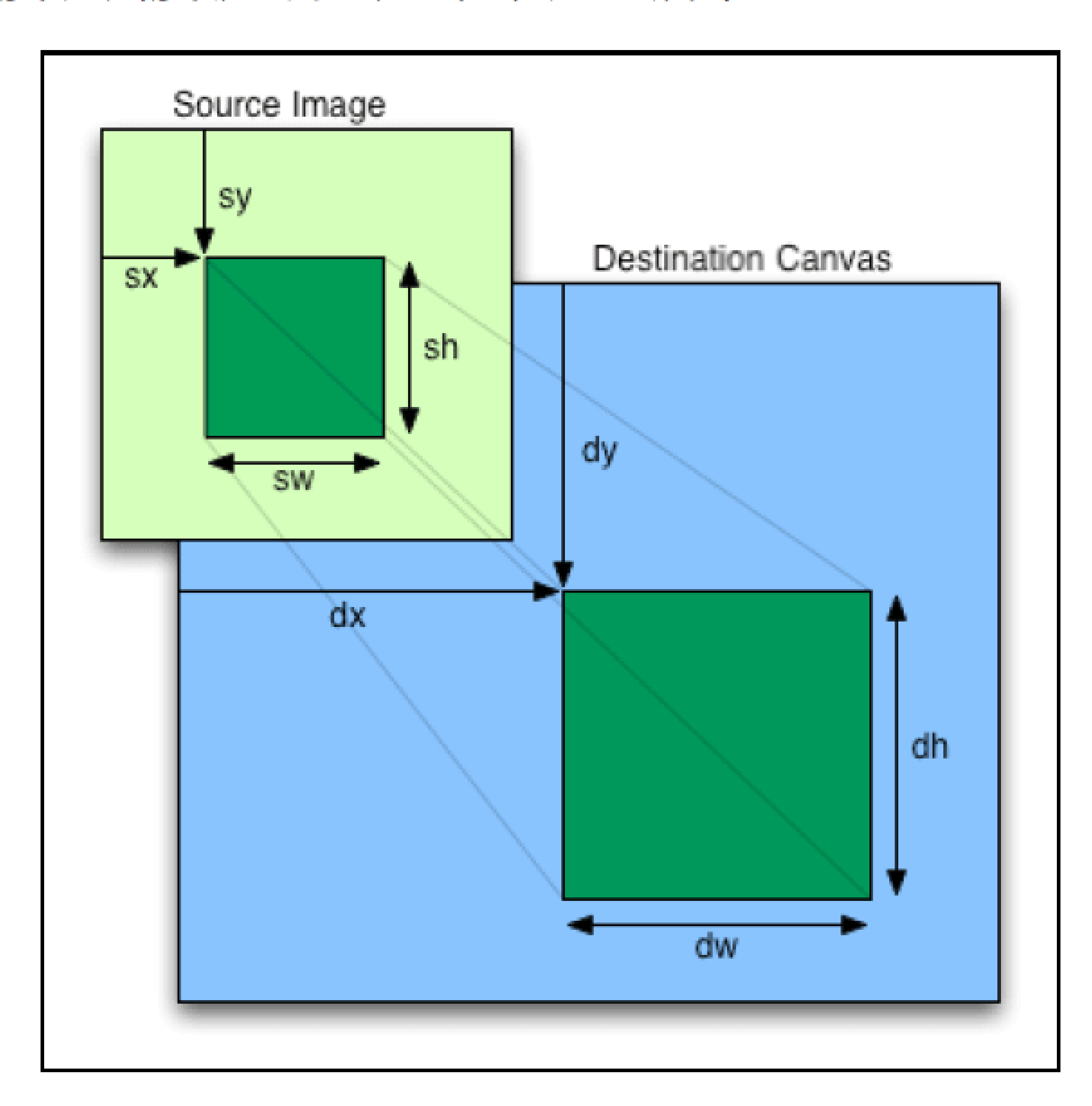


图 4-6　裁剪图像

canvas 的 clip 方法也可以实现图像的剪裁，需要结合画布路径一起使用。首先创建路径，然后调用 clip 方法设置剪裁区域，绘制图像时则只绘制路径所包括区域内的图像。例如：

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<script type="text/javascript">
window.onload = function ()
{
    var c=document.getElementById("myCanvas");
    var ctx=c.getContext("2d");
    var image = new Image();
    image.src = "tul.jpg";
    image.onload = function ()
    {
        ctx.beginPath();
```

```
            ctx.arc(150,150,75,0,Math.PI*2,true);
            ctx.closePath();
            ctx.clip();
            ctx.drawImage(image, 0, 0, image.width, image.height);
        }
    }
</script>
</head>
<body>
<p>画布：</p>
<canvas id="myCanvas" width="700" height="300" style="border:1px solid
    #d3d3d3;background:#ffffff;">
    Your browser does not support the HTML5 canvas tag.
</canvas>
</body>
</html>
```

代码运行效果如图 4-7 所示。



图 4-7　clip 方法的运行效果

像素的处理

canvas 提供了三个方法用于像素级操作：createImageData、getImageData 和 putImageData。createImageData 方法的语法为：

```
var imgData=context.createImageData(width,height);
```

或者

```
var imgData=context.createImageData(imageData);
```

前者以指定的尺寸(以像素计)创建新的 ImageData 对象，后者则创建与另一个 ImageData 对象尺寸相同的新 ImageData 对象(不会复制图像数据)。

createImageData 返回的 ImageData 对象保存了图像像素值，它有三个属性：width、height 和 data。data 属性的类型为 Uint8ClampedArray，用于储存 width*height*4 个像素值。每一个像素有 RGB 值和透明度 alpha 值。为了更好地理解其原理，让我们来看一个例子——绘制绿色矩形。

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<script type="text/javascript">
window.onload = function ()
{
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
var imgd = ctx.createImageData(50,50);
var pix = imgd.data;
for (var i = 0; n = pix.length, i < n; i += 4)
 {
  pix[i] = 255;
  pix[i+1]=0;
  pix[i+2]=0;
  pix[i+3] = 127;
}
ctx.putImageData(imgd, 0,0);
}
</script>
</head>
<body>
<p>画布：</p>
<canvas id="myCanvas" width="300" height="300" style="border:1px solid
    #d3d3d3;background:#ffffff;">
    Your browser does not support the HTML5 canvas tag.
</canvas>
</body>
</html>
```

ImageData 的内容类似于[r1,g1,b1,a1,r2,g2,b2,a2,r3,b3.g3,a3,...]，其中，r1,g1,b1,a1 为第一个像素的红色值、绿色值、蓝色值、透明值；r2,g2,b2,a2 为第二个像素的红色值、绿色值、蓝色值、透明值；以此类推。

在画布中访问像素的方法是 getImageData。语法如下所示：

```
context.getImageData(x, y, width, height);
```

这个方法有 4 个参数：要访问的像素区域的原点坐标(x，y)、像素区域的宽度和高度，返回一个 ImageData 对象。

putImageData() 方法将图像数据(从指定的 ImageData 对象)放回画布上。语法如下所示：

```
context.putImageData(imgData,x,y[,dirtyX,dirtyY,dirtyWidth,dirtyHeight])
```

这个方法有 7 个参数：imgData 为像素数组；x、y 分别表示绘制图形的起点横坐标和纵坐标；dirtyX、dirtyY、dirtyWidth、dirtyHeight 为可选参数，表示一个矩形，如果提供这 4 个参数，则只会绘制这个矩形范围内的图像。例如：

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<script>
window.onload = function ()
{
    var c=document.getElementById("myCanvas");
    var ctx=c.getContext("2d");
    ctx.fillStyle="green";
    ctx.fillRect(10,10,140,140);
}
function copy()
{
    var c=document.getElementById("myCanvas");
    var ctx=c.getContext("2d");
    var imgData=ctx.getImageData(10,10,140,140);
    ctx.putImageData(imgData,10,160,0,0,70,70);
}
</script>
</head>
<body>
<p>画布：</p>
<canvas id="myCanvas" width="300" height="300" style="border:1px solid
    #d3d3d3;background:#ffffff;">
    Your browser does not support the HTML5 canvas tag.
</canvas>
<button onclick="copy()">复制</button>
</body>
</html>
```

代码运行效果如图 4-8 所示。

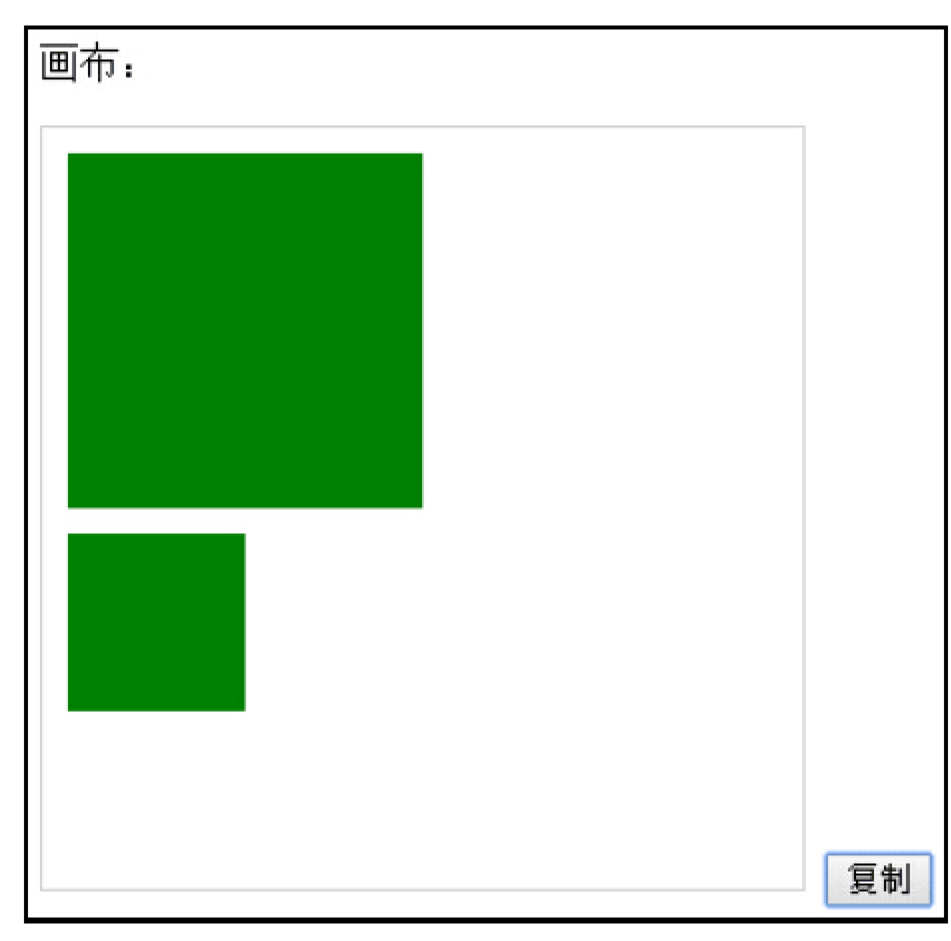


图 4-8　putImageData()方法的运行效果

本章小结

本章介绍了弹力球游戏的设计过程，并详细介绍了游戏制作中通用的代码结构。学习了 canvas 如何绘制线条、矩形、圆形和文本。本章 canvas 的内容比较基础，希望这些内容能够拓宽我们的眼界，帮助我们进一步学习画布的高级功能。

本章练习

1. canvas 的用途是什么？
2. 用 canvas 绘制三条不同的竖线。

第5章　MongoDB

2009 年，NoSQL 概念被提了出来，那时的 NoSQL 主要是指非关系型、分布式、不提供数据库的设计模式。MongoDB 是目前非常流行的非关系数据库，本章我们将进入 MongoDB 的世界。

本章内容：

- 如何下载及安装 MongoDB
- 掌握 MongoDB 的文档和集合概念
- 如何使用 MongoDB 索引

5.1　MongoDB初探

MongoDB 是一个开源的、面向文档存储的数据库，是 NoSQL 数据库的一种。NoSQL 最常见的解释是 Non-Relational(非关系型)，Not Only SQL(不仅是 SQL)也被很多人接受。

NoSQL 用得最多的当数 key-value 存储，当然还有其他的文档型的列存储、图形数据库、XML 数据库等。在 NoSQL 概念提出之前，这些数据库就被用于各种系统当中，但是却很少用于 Web 互联网应用。随着 Web 2.0 的快速发展，非关系型、分布式数据存储得到了快速发展，它们不保证关系型数据库的 ACID(原子性、一致性、隔离性、持久性)特性。MongoDB 的名称取自 humongous(巨大的)的中间部分，足见 MongoDB 的宗旨。在处理大量数据上面，MongoDB 是可扩展、高性能的下一代数据库，由 C++语言编写，旨在为 Web 应用提供可扩展的高性能数据存储解决方案。它的特点是高性能、易部署、易使用，存储数据非常方便，主要特性有：

- 模式自由，支持动态查询、索引，可轻易查询文档中内嵌的对象及数组。
- 面向文档存储，易存储对象类型的数据，包括文档内嵌对象及数组。
- 高效的数据存储，支持二进制数据及大型对象(如照片和视频)。
- 支持复制和故障恢复，提供了主-从、主-主模式的数据复制及服务器之间的数据复制。
- 自动分片以支持云级别的伸缩性，支持水平的数据库集群，可动态添加额外的服务器。

与传统关系型数据库相比，MongoDB 也有劣势，它不适用于要求高度事务性的系统、传统的商业智能系统、复杂的跨文档(表)级联查询。而对于大数据量、高并发、弱事务的互联网应用，MongoDB 可以应对自如。

本章将使用 MongoDB 完成人人微博网站的数据库设计。数据库的设计共分为三个部分：用户表结构设计、微博表结构设计、粉丝关注表结构设计。

5.2　MongoDB 的下载与安装

MongoDB 的官方下载地址为 http://www.MongoDB.org/downloads。我们这里在 Windows 64 位机器上安装 MongoDB，选择的版本为 Windows 64-bit 2008 R2+非 SSL 安全协议版本，下载步骤如图 5-1 所示。

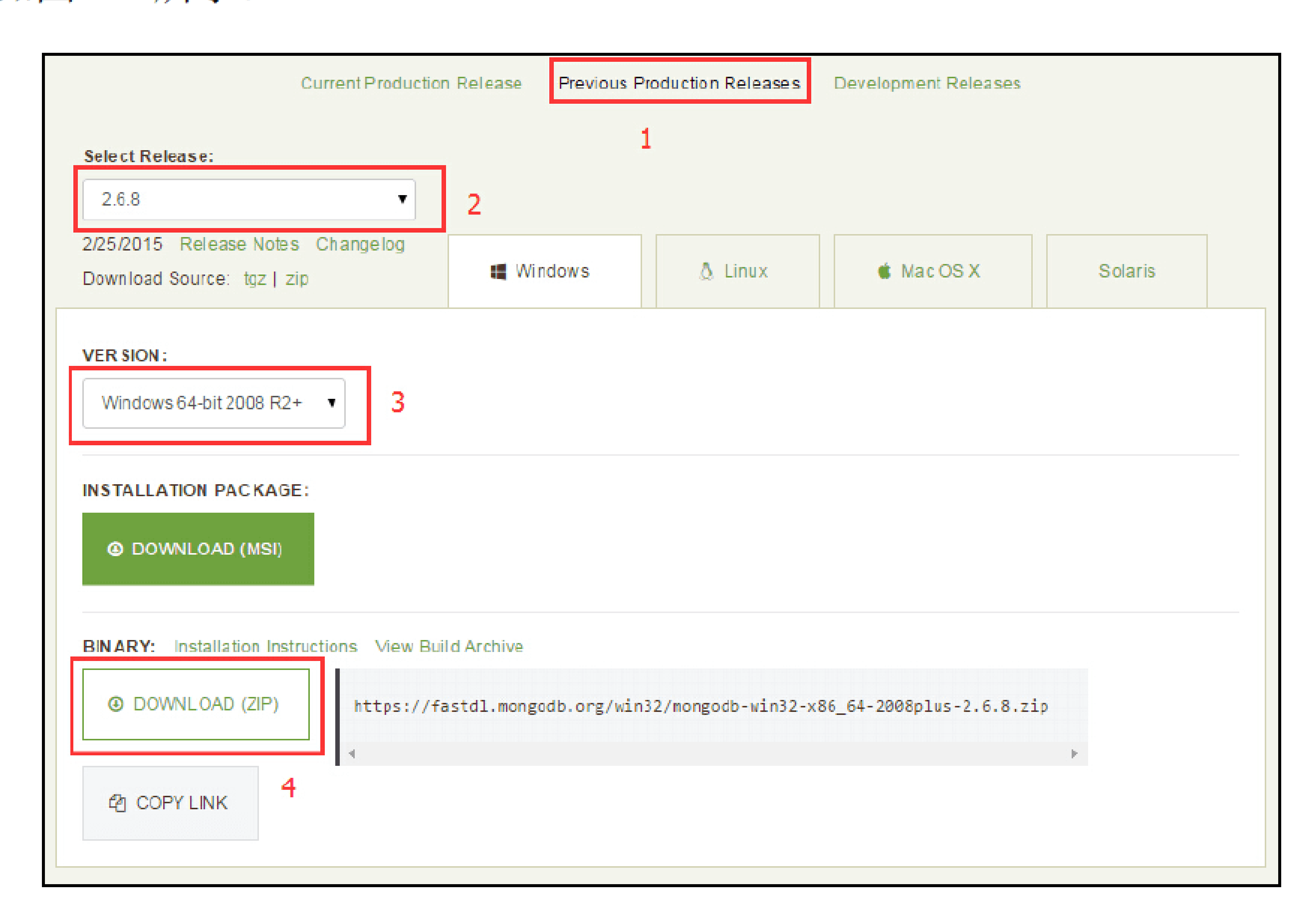


图 5-1　MongoDB 下载步骤

下载后的文件名为 MongoDB-win32-x86_64-2008plus-2.6.8.zip，将其解压到 G 盘根目录下。安装完毕后的 MongoDB 只有十几个 exe 文件，如图 5-2 所示。

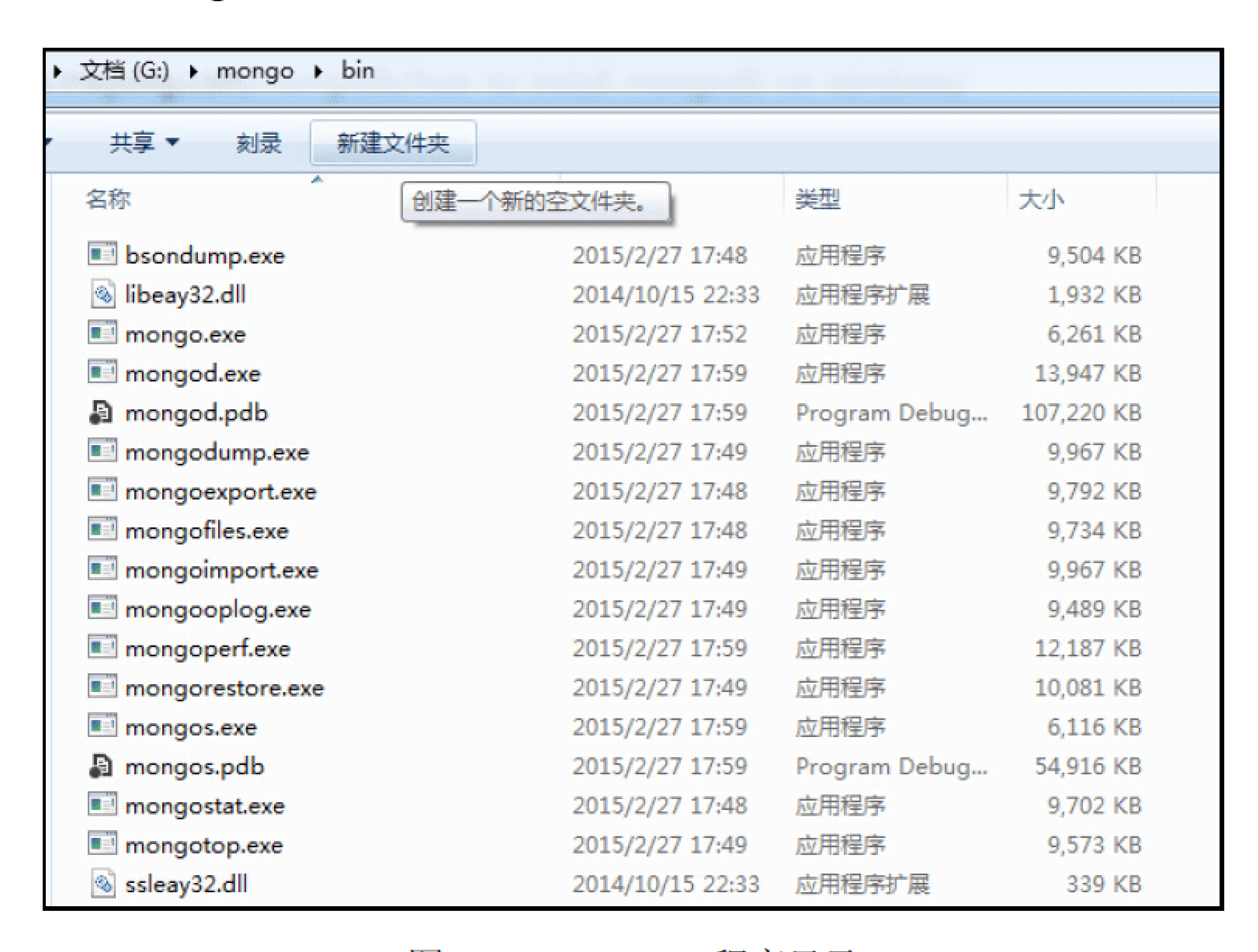


图 5-2　MongoDB 程序目录

MongoDB 的数据目录不会主动创建，我们在安装完成后需要创建它。下面创建一个新目录：G:\MongoDB-win32-x86_64-2008plus-2.6.8\data。

在命令行中输入 mongod --dbpath=..\data 运行 MongoDB，默认端口是 27017，如图 5-3 所示。

```
命令提示符 - mongod --dbpath=G:\mongo\data
G:\mongo\bin>mongod --dbpath=G:\mongo\data
2015-03-06T13:48:03.532+0800 I CONTROL  Hotfix KB2731284 or later update is not
installed, will zero-out data files
2015-03-06T13:48:03.533+0800 I CONTROL  [initandlisten] MongoDB starting : pid=5
524 port=27017 dbpath=G:\mongo\data 64-bit host=zhangpeng-PC
```

图 5-3　启动 MongoDB

表 5-1 为 MongoDB 的启动参数说明。

表 5-1　MongoDB 启动参数

参　　数	说　　明
--bind_ip	绑定服务 IP，若绑定 127.0.0.1，则只能本机访问
--logpath	指定 MongoDB 日志文件，注意是指定文件，不是指定目录
--logappend	使用追加的方式写日志
--dbpath	指定数据库路径
--port	指定服务端口号，默认端口是 27017
--serviceName	指定服务名称
--serviceDisplayNam	指定服务名称，有多个 MongoDB 服务时执行
--install	指定作为 Windows 服务安装

MongoDB 数据库启动成功后，就可以使用 mongo.exe 连接数据库，如图 5-4 所示。

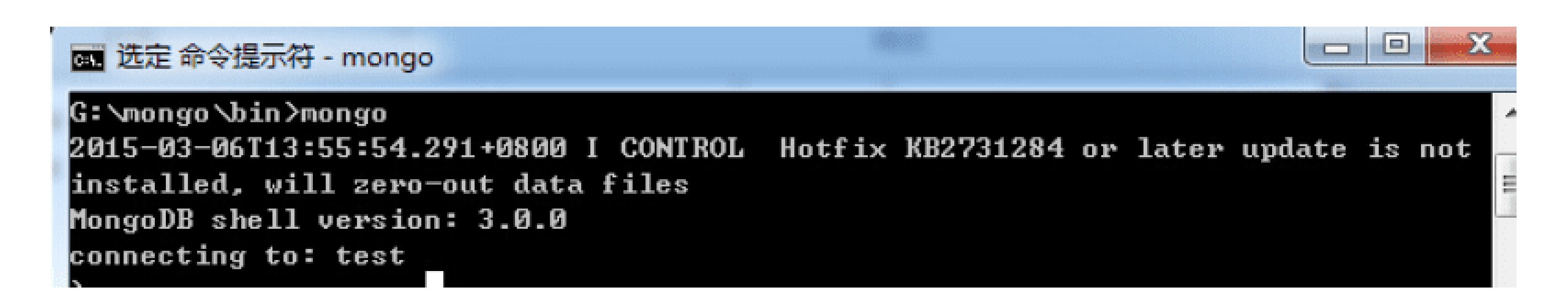


图 5-4　连接 MongoDB

mongo 命令提供了操作数据库的一系列 API，并且是使用 JavaScript 编写的 API，那么我们就可以使用 JavaScript 的所有内置对象。而其他操作命令可以通过 help 命令学习，如图 5-5 所示。

```
> help
        db.help()                    help on db methods
        db.mycoll.help()             help on collection methods
        sh.help()                    sharding helpers
        rs.help()                    replica set helpers
        help admin                   administrative help
        help connect                 connecting to a db help
        help keys                    key shortcuts
        help misc                    misc things to know
        help mr                      mapreduce

        show dbs                     show database names
        show collections             show collections in current database
        show users                   show users in current database
        show profile                 show most recent system.profile entries wit
h time >= 1ms
        show logs                    show the accessible logger names
        show log [name]              prints out the last segment of log in memor
y, 'global' is default
        use <db_name>                set current database
        db.foo.find()                list objects in collection foo
        db.foo.find( { a : 1 } )     list objects in foo where a == 1
        it                           result of the last line evaluated; use to f
urther iterate
        DBQuery.shellBatchSize = x   set default number of items to display on s
hell
        exit                         quit the mongo shell
>
```

图 5-5　help 命令

5.3　MongoDB 的基本概念

5.3.1　数据库

一个 MongoDB 中可以建立多个数据库。MongoDB 中，show dbs 命令可以显示所有数据的列，执行 db 命令可以显示当前数据库对象，运行 use 命令可以连接到指定的数据库。例如：

```
db;
show dbs;
use mydb;
```

运行效果如图 5-6 所示。

```
> db
test
> show dbs;
admin   (empty)
local   0.078GB
test    0.078GB
> use mydb;
switched to db mydb
```

图 5-6　数据库操作

因为每个数据库对应文件系统中的文件，所以每个数据库都有独立权限。

5.3.2　文档和集合

文档是 MongoDB 中数据的基本单元，非常类似于关系数据库管理系统中的行。但两者并不完全对等。MongoDB 以类 JSON 的格式来表示文档。例如：

```
{'title':'my blog title', 'content':'my blog content', 'date':new Date()}
```

这种格式有一个固定的名字——BSON，BSON(Binary Serialized Document Format)是一

种类 JSON 的二进制形式的存储格式，简称 Binary JSON。它和 JSON 一样，支持内嵌的文档对象和数组对象，但是 BSON 有 JSON 没有的一些数据类型。

表 5-2 为 BSON 中常用的几种数据类型。

表 5-2　BSON 数据类型

类　型	说　明
null	用于表示空值或不存在的字段
32 位整型	JavaScript 中只有一种数字类型，而 MongoDB 中有 3 种类型，默认情况下 shell 的数字都被 MongoDB 当作双精度。在 shell 下修改文档中的 32 位整型，也会被转换为 64 位浮点数。MongoDB 提供了 NumberLong 对象来包装 64 位整型。所以尽量不要在 shell 下修改文档
64 位整型	
64 位浮点数	
日期	日期类型存储从标准纪元开始的毫秒数
正则表达式	采用 JavaScript 的正则表达式语法
字符串	字符串
布尔型	布尔型
二进制数据	二进制数据
数组	值的集合表示为数组。同 JSON 数组一样，数组是一组值，既可以作为有序对象(队列、栈等)来操作，也可以作为无序对象来操作。数组中可以包含不同数据类型的对象，甚至是嵌套数组
内嵌文档	文档可以包含别的文档，也可以嵌入到父文档中。内嵌文档就是把整个 MongoDB 文档作为另一个文档中键的值。与数组一样，MongoDB 能够理解内嵌文档的结构，并构建索引、执行查询、进行更新等
ObjectId	在 MongoDB 中的文档需要使用唯一的关键字_id 来标识它们。几乎每一个 MongoDB 文档都使用_id 字段作为第一个属性(在系统集合和固定容量集合(capped collection)中有一些例外)。_id 值可以是任何类型，最常见的做法是使用 ObjectId 类型

ObjectId 是_id 的默认类型，是全局唯一的，由 12 个字节组成，每个字节是两位十六进制数字，总共是一个 24 位的字符串。其中 0～3 字节是时间戳(单位秒)，4～6 字节是主机名散列值，7 和 8 字节是 PID(进程标识符，确保每个进程的 ObjectId 不同)，9～11 字节是计数器(确保同一进程的同一秒产生的 ObjectId 不同，最大值为 256 的 3 次方)。

集合就是一组文档的组合。如果将文档类比成数据库中的行，那么集合就可以类比成数据库中的表。可使用 show collections 显示当前数据库中的集合。

MongoDB 中的集合是无模式的，关系型数据库中表的结构是固定的，但是 MongoDB 中集合存储的文档的结构可以是不同的，比如下面的两个文档可以同时存入一个集合中：

```
{"name":"mengxiangyue"} {"Name":"mengxiangyue","gender":"male"}
```

MongoDB 中有一种特殊类型的集合——固定集合(Capped Collection)，也就是固定大小的集合，在创建时要预先指定大小。如果空间用完，新添加的对象将会取代集合中最旧的对象，以永远保持最新的数据。它自动维护插入顺序，在某些特殊的使用场景中非常有效，如记录日志的场景，某监控软件只保留一个月的监控数据可查。例如：

```
db.createCollection( "log", { capped: true, size: 12, max: 50, autoIndexId:
    false } )
```

capped：是否启用集合限制，如果开启则需要指定 size 值。

size：限制集合使用空间的大小，如果指定的值小于 4096，则集合的空间大小为 4096 字节，如果大于 4096 则会分配相近的 256 整数倍大小的空间。

max：集合中的最大条数限制，默认为没有限制。

autoIndexId：是否使用_id 作为索引，默认为使用，值为 true。

下面的例子进行批量插入数据：

```
for (i=1;i<100;i++) {
db.log.insert({'loginfo' : 'loginfo','loglevel' : 'loglevel','insertid' : i})
}
db.log.count();//输出 37
```

通过调用 count 函数得到的输出结果为 37，而我们插入了 99 条数据，可见插入的数据已经超过了 4096 字节。

如果需要删除一个集合，可以使用 drop()函数，例如：

```
db.log.drop();//输出 true
```

我们已经对 MongoDB 的基本概念有所了解，下面来设计人人微博网站的数据库。在第 2 章中我们分析了用户个人信息界面的需求，这里参考表 2-1 的分析结果，对用户表结构设计如下(见表 5-3)。

表 5-3　users 文档结构

字　段	说　明	类　型	备　注
_id	用户 id	数字类型	
login_name	登录名	String	唯一
nick_name	昵称	String	唯一
img	缩略图地址	String	
real_name	真实姓名	String	
provinceValue	所在省份编号	String	
provinceName	所在省份名称	String	
cityValue	所在城市编号	String	
cityName	所在城市名称	String	
gender	性别	String	只有 male 和 female 两个可选值
age	年龄	String	
other_email	备用邮箱	String	
blog	博客地址	String	
qq	QQ	String	
pwd	密码	String	

(续表)

字 段	说 明	类 型	备 注
msn	MSN	String	MSN
birthday	生日	String	
status	记录是否有效	布尔型	
timestamp	创建时间	NumberLong，从 1970-01-01 00:00:00 UTC 到现在的毫秒数	

发布的微博应该包含文字信息、图片信息、发布日期等，下面是微博内容的表结构设计(见表 5-4)。

表 5-4 weibos 文档结构

字 段	说 明	类 型	备 注
_id	主键 id	数字类型	
user_id	发布者 id	数字类型	非空
user_nickName	发布者昵称	String	
user_head_img	发布者缩略图地址	String	
img	微博图片地址	String	
content	微博内容	String	非空
status	记录是否有效	布尔型	
timestamp	微博创建时间	NumberLong，从 1970-01-01 00:00:00 UTC 到现在的毫秒数	

用户之间可以互相关注，用户和粉丝之间的关系用 myfans 文档保存，下面是 myfans 文档结构(见表 5-5)。

表 5-5 myfans 文档结构

字 段	说 明	类 型	备 注
_id	主键 id	数字类型	
followed_id	被关注用户 id	数字类型	
fans_id	粉丝用户 id	数字类型	
status	记录是否有效	布尔型	
timestamp	创建时间	NumberLong，从 1970-01-01 00:00:00 UTC 到现在的毫秒数	

5.4　MongoDB 常用命令

5.4.1　插入数据

当我们向某集合中插入文档时，如果该集合不存在，MongoDB 会自动创建该集合，不需要手动创建集合，同时也不需要指定文档的结构。例如：

```
j = {name:"mongo"}
k = {x:3}
db.users.insert(j)
db.users.insert(k)
```

文档必须有一个_id 键。这个键的值可以是任何类型，默认是 ObjectId 对象。 如果插入文档的时候没有_id 键，系统会自动创建一个。如果我们自己指定_id 的值，要明确_id 键的类型，不然如果我们保存_id 键时使用字符串类型，查询时使用数字类型，就查不到数据。

5.4.2　查询数据

MongoDB 最大的功能之一就是它支持动态查询，这与传统的关系型数据库查询一样，但是它的查询更灵活。

1. 查询所有记录

最简单的查询是查询集合中的所有记录，例如：

```
db.users.find()
db.users.find({})
```

这里，表达式对象是一个空文档，在查询的时候匹配所有的记录。再看下面的代码：

```
db.users.find({'last_name': 'Smith'})
```

这里，我们将查询出所有 last_name 属性值为 Smith 的文档记录。

2. 查询特定字段值

MongoDB 还支持一些额外的参数选项。例如，我们可能只想返回某些特定的字段值。

```
//返回除了 pwd 字段外的所有字段
 db.users.find({},{pwd:0});
//返回 login_name 等于 weibo1@163.com 除了 pwd 的所有列
db. users.find( { login_name : 'weibo1@163.com' }, { pwd : 0 } );
 //只返回 login_name 字段
db.user.find({_id:6},{login_name:1});
```

3. 查询表达式

在查询文档时，还可以使用条件表达式，使用 or 表达式的例子为：

```
db.users.find({ '$or' : [{'name' : 'hurry'}, {'age' : 18}] },{'name' : 1, 'age' :
    1, 'skills' : 1});
```

这相当于关系型数据库中的：

```
select name, age, skills from users where name = 'hurry' or age = 18;
```

$ne 的意思是 not equal(不等于)，例如：

```
//查询_id 不等于 1 的文档
db.user.find({_id:{$ne:1}})
```

我们还可以使用大于($gt)、小于($lt)、大于等于($gte)、小于等于($lte)表达式，例如：

```
//查询 age 大于等于 20 小于等于 30 的文档
db.users.find({'age' : {'$gte' : 20, '$lte' : 30}});
```

还有其他一些表达式，如表 5-6 所示。

表 5-6　查询表达式

表 达 式	例　　子	对应关系型数据库 sql 语句
$in、$nin	db.users.find({'age' : {'$in' : [10, 22, 26]}});	select * from users where age in (10, 22, 26);
匹配 null	db.users.find({'age' : null});	select * from users where age is null;
like(MongoDB 支持正则表达式)	db.users.find({name:/hurry/});	select * from users where name like "%hurry%";
distinct	db.users.distinct('name');	select distinct (name) from users;
count	db.users.count();	select count(*) from users;

4. 分页查询

MongoDB 还支持使用 skip 和 limit 命令来进行分页查询，例如：

```
//跳过前 10 条记录
 db.users.find().skip(10);
//每页返回 8 条记录
db.users.find().limit(8);
//跳过前 20 条记录，并且每页返回 10 条记录
db.users.find().skip(20).limit(8);
```

5. 对结果进行排序

```
//先按照_id 升序排列，再按 login_name 降序排序
db.users.find().sort({_id:1,login_name:-1});
```

其中 1 表示升序，-1 表示降序。

5.4.3 删除数据

下面是删除数据的示例：

```
//删除_id 为 2 的数据
db.users.remove({"id":2});
//删除所有数据
db.users.remove();
```

5.4.4 更新数据

MongoDB 最常用的更新命令是 update 命令，它的语法为：

```
db.collection.update( criteria, objNew, upsert, multi )
```

criteria：update 的查询条件，类似 SQL 语句 update 内 where 后面的部分。

objNew：update 的对象和一些更新的操作符(如$set 和$inc)等，也可以理解为 SQL 语句 update 内 set 后面的部分。

upsert：这个参数的意思是如果不存在 update 的记录，是否插入 objNew，true 为插入，false 为不插入。默认是 false。

multi：默认是 false，只更新找到的第一条记录，如果这个参数为 true，就把按条件查出的多条记录全部更新。

$set 的语法为：

```
{ $set : { field : value } }
```

相当于 SQL 的 set field = value，全部数据类型都支持$set。例如：

```
db.users.find( { "_id" : 2 } );
//输出{ "_id" : 2, "login_name" : "weibo1@163.com", "gender" : "male",
//"nick_name" : "111111", "pwd" : "111111", "status" : true, "timestamp" :
//NumberLong("1425865794104") }

db.users.update( { "_id" : 2 } , { $set : { "login_name" : " weibo2@163.com "} } );
// 输出 WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.users.find( { "_id" : 2 } );
//输出{ "_id" : 2, "login_name" : "weibo2@163.com", "gender" : "male",
//"nick_name" : "111111", "pwd" : "111111", "status" : true, "timestamp" :
//NumberLong("1425865794104") }
```

$inc 的语法为：

```
{ $inc : { field : value } }
```

意思是对数字字段 field 增加 value，例如：

```
db.users.find( { "_id" : 2 } );
//输出  { "_id" : 2, "login_name" : " weibo2@163.com ", "gender" : "male",
//"nick_name" : "111111", "pwd" : "111111", "status" : true, "timestamp" :
//NumberLong("1425865794104"), "login_name" : " weibo2@163.com ", "age" :
//10000 }

 db.users.update( { "_id" : 2 } , { $inc : { "age" : -9000 } } );

db.users.find( { "_id" : 2 } );
//输出{ "_id" : 2, "login_name" : " weibo2@163.com ", "gender" : "male",
//"nick_name" : "111111", "pwd" : "111111", "status" : true, "timestamp" :
//NumberLong("1425865794104"), "login_name" : " weibo2@163.com ", "age" :
//1000 }
```

5.4.5　索引

索引是一种提高查询速度的机制，数据库索引有些像书籍的目录，有了目录就不需要翻遍整本书。索引在数据结构上可以分为 B 树索引、位图索引和散列索引。MongoDB 使用的是 B 树索引。根据不同的应用需求，还可以分成唯一索引(unique)、稀疏索引(sparse)等几种类型。

可使用 createIndex 创建索引，例如：

```
db.users.createIndex({_id: 1})
```

索引的方向为 1 或-1：1 表示升序，-1 表示降序。

1. 唯一索引

唯一索引在创建时加上 unique:true 选项即可，创建命令如下：

```
db.users.createIndex({_id: 1}, {unique: true})
```

上面的唯一索引创建后，如果 insert 一条_id 已经存在的数据，则会报如下错误：

```
E11000 duplicate key error index
```

如果我们在一个已有数据的 collection 上创建唯一索引，若唯一索引对应的字段原来就有重复的数据项，那么创建会失败，需要加上一个 dropDups 选项来强制将重复的项删除，命令如下：

```
db.users.createIndex ({age: 1}, {unique: true, dropDups: true})
```

上面的语句有一点需要注意，如果有的文档上没有 age 字段，而我们又使用了 dropDups: true，那么创建唯一索引时 MongoDB 将删除没有 age 字段的文档。如果配合使用即将讲到的 sparse:true 选项，数据库将保留没有 age 字段的文档。代码如下：

```
db.users.createIndex ({age: 1}, {unique: true, dropDups: true, sparse: true })
```

2. 松散索引

当数据中没有某个字段或字段值为 null 时，如果在这个字段上建立普通索引，那么无此字段或此字段值为 null 的文档也会参与到索引结构中，占用相应的空间。如果我们不希望这些值参与到索引中，可以采用松散索引，松散索引只会让指定字段不为空的行参与到索引创建中来。创建一个松散索引可以用下面的命令：

```
db.users.createIndex ({age: 1}, {sparse: true})
```

3. 后台创建索引

创建索引时数据库会阻塞，如果希望在索引创建的过程中，数据库还可以正常地读写，可以指定 background 为 true，默认值是 false。例如：

```
db.people.createIndex( { zipcode: 1}, {background: true} )
```

4. 为排序创建索引

在查询数据的时候经常对结果集进行排序，这时候就会使用索引，比如我们创建的索引：

```
db.records.createIndex( { a: 1 } )
```

在排序中使用索引能够提升查询的性能，上面的语句能够支持如下对 a 字段的排序：

```
db.records.find().sort( { a: 1 } )
db.records.find().sort( { a: -1 } )
```

用户可以在多个列上建立索引，这种索引叫做复合索引(组合索引)。如果使用了复合索引，对结果集进行排序的时候需要注意，比如：

```
db.records.createIndex( { a: 1, b: 1 } )
```

它能够支持的排序有{ a: 1, b: 1}，但不支持{ b: 1, a: 1}，使用复合索引排序时需要指定相同的排序方向或完全相反的排序方向，上面的语句还支持对{ a: -1, b: -1}排序，但不支持对{ a: -1, b: 1}排序。

一般来说，如果索引包含 N 个键，则对于前几个键的查询会有帮助。比如有个索引{"a":1,"b":1,"c":1,"z":1}，实际上有了{"a":1}、{"a":1,"b":1}、{"a":1,"b":1,"c":1}等索引。但是使用{"b":1}、{"a":1,"c":1}等的排序(或查询)则不会优化，只有使用索引前部的排序(或查询)才能使用该索引。

5. 查询索引

可以使用 getIndexes 方法查询集合上的所有索引，例如：

```
db.users.getIndexes()
```

运行结果如图 5-7 所示。

```
> db.users.getIndexes()
[
        {
                "v" : 1,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "db_main.users"
        },
        {
                "v" : 1,
                "unique" : true,
                "key" : {
                        "age" : 1
                },
                "name" : "age_1",
                "ns" : "db_main.users",
                "dropDups" : true,
                "sparse" : true
        }
]
```

图 5-7　查询索引

6. 删除索引

dropIndex()方法可以删除索引，它的参数可以是索引名称，也可以是创建索引时的文档格式。例如：

```
db.users.dropIndex( "age_1" )
db.users.dropIndex( { "age" : 1 } )
```

人人微博网站中 users 文档的登录名和昵称要求唯一，myfans 文档的 followed_id 和 fans_id 组合唯一，下面是实现代码：

```
db.myfans.createIndex({'followed_id': 1,'fans_id':1} , {unique: true,
    dropDups: true })
db.users.createIndex({'login_name': 1} , {unique: true, dropDups: true })
```

本章小结

本章我们学习了 MongoDB 的基本概念，重点介绍了查询语句和索引。

索引也有一些缺点，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加，索引需要占用物理空间。如果要建立聚簇索引，那么需要的空间就会更大。当对表中的数据进行增加、删除和修改的时候，索引也要动态维护，这样就降低了数据的维护速度。如果查询要返回集合中一半以上的结果，则用表扫描比查询索引要高效一些。所以在创建索引时应该考虑会做什么样的查询，还要考虑哪些键需要索引以及如何使得创建的索引具有可扩展性。

本章练习

1. MongoDB 的文档格式是什么？
2. MongoDB 的适用场景是什么？

第6章　jQuery

为了帮助开发人员构建 Web 应用，出现了被称为库或框架的 JavaScript 代码集合。jQuery 是目前世界上 JavaScript 领域最受欢迎的库，它能够大大提高编程的效率。

本章内容：

- jQuery 是什么及其作用
- 使用 jQuery 处理 DOM
- 使用 jQuery 处理事件

6.1　jQuery 入门

6.1.1　jQuery 初体验

jQuery 是一个快速、体积小、功能丰富的 JavaScript 库。它通过一个跨多种浏览器的易于使用的 API 使得很多事情变得更容易，例如 HTML 文档遍历和操作、事件处理、动画效果以及 Ajax 交互。它兼容各种浏览器，比如 IE 浏览器、Opera 浏览器等，但是在 2.0 版本以及后续版本中将不再支持 IE8 及更新的版本。

与 jQuery 类似的 JavaScript 框架有 Prototype、Yui、Ext、Dojo 等，随着 jQuery 的不断发展，jQuery 已经成为最流行的 JavaScript 框架，全世界访问量最大的前 10 000 个网站中，有一半甚至一半以上都在使用 jQuery。

jQuery 的理念是"write less，do more"，它对许多丰富的程序功能进行了封装，供其他人调用，使得 Web 程序开发变得简洁并能提高开发效率。jQuery 的语法通俗易懂，这也是它能够流行的原因。例如，下面是一行有效的 jQuery 代码：

```
$( "#footer" ).hide( "slow" ).addClass( "foo" ).show( "fast" )
```

我们来猜一下上面的代码是做什么的？第一部分$("#footer")猜不出来，但是剩余的部分很容易理解：

- hide()方法会隐藏一些东西
- addClass()方法会添加一个 CSS 类
- show()方法显示一些东西

本章的内容将会讲解这些代码如何运行及其实现的功能。

6.1.2　jQuery 环境搭建

jQuery 的环境搭建非常简单，我们只需要下载 jQuery 并在 HTML 页面中引入 jQuery 就可以了。jQuery 的官方网站(http://jQuery.com/)提供了最新的 jQuery 框架下载链接，如图 6-1 所示。

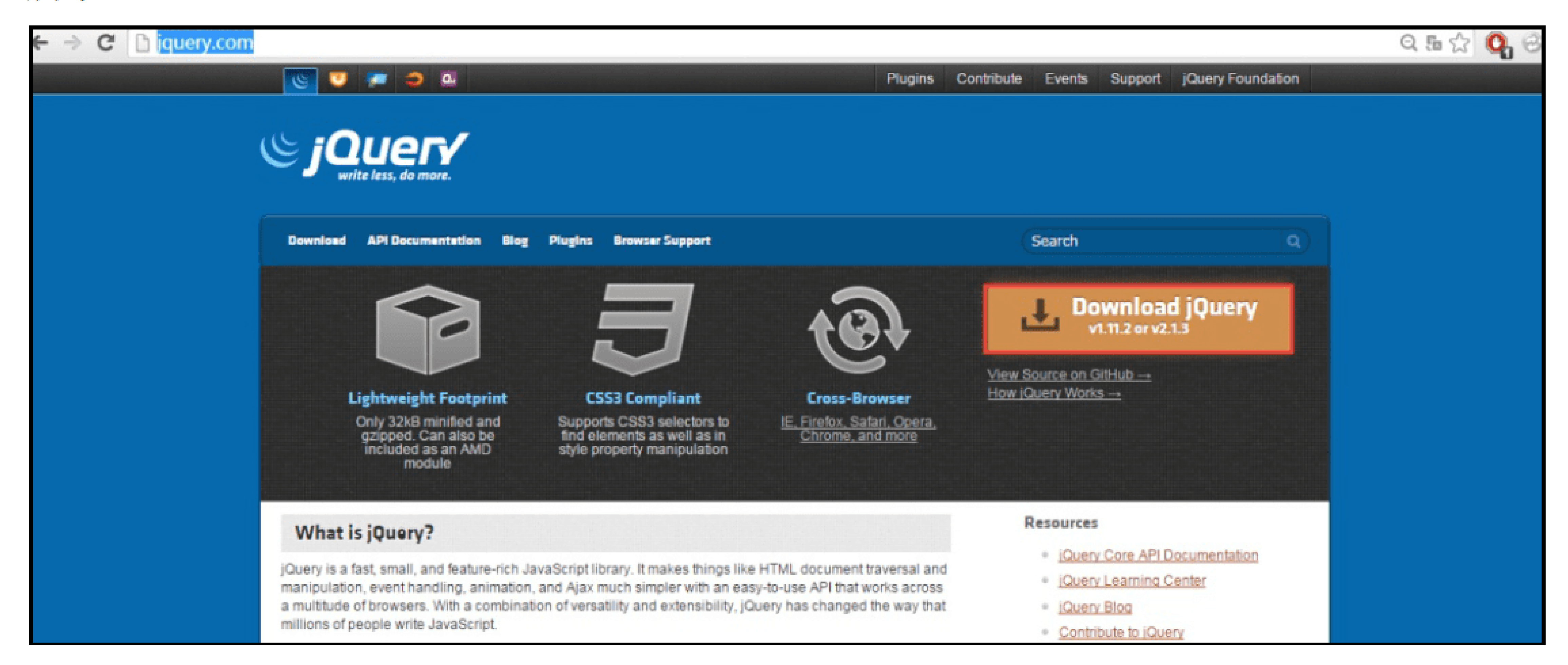


图 6-1　jQuery 官网

jQuery 分为压缩版本(min 版本)和非压缩版本。min 版本主要应用于已经开发完成的网页中，也就是生产环境当中。而非压缩版本的文件比较大，里面有整洁的代码书写规范和注释，主要应用于网页开发过程当中。

使用时直接将该 js 文件用<script>标签引入到自己的页面中即可，代码如下：

```
<script src = "jQuery-2.1.1.js" ></script>
```

另一种使用 jQuery 的方案是，使用由内容分发网络(CDN)所提供的一个公共版本。CDN 是分布于世界各地的快速服务器，设计用于尽可能以最快的速度提供文件服务。其基本思路是尽可能避开互联网上有可能影响数据传输速率和稳定性的瓶颈和环节，使内容传输得更快、更稳定。通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络，CDN 系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决互联网拥挤的状况，提高用户访问网站的响应速度。对互联网感兴趣的大型公司(如 Google、百度)提供了免费的各种版本的 jQuery。我们只需要知道由 CDN 服务提供的 URL 就可以了，例如：

```
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<script src = "https://code.jQuery.com/jQuery-2.1.1.js" ></script>
```

6.1.3　jQuery 中的$及其作用

下面是一个最简单的 jQuery 程序：

```
<script>
    $(document).ready(function() {
        alert("Hello World");
    });
</script>
```

jQuery 提供了几个核心对象，而最频繁使用的是 jQuery 对象。它的语法是：

```
jQuery(elements)
```

elements 表示一个或多个 DOM 元素，返回一个 jQuery 对象。为了编写代码的方便，使用美元符“$”来代替“jQuery”。在使用 jQuery 编程的上下文环境中，两者是等价的。$最常用的场景是用作选择器接收一个表示 CSS 选择器的字符串，并返回匹配该字符串的一个或多个 HTML 元素。如果对 CSS 选择器比较熟悉，就比较容易学习 jQuery 的选择器。例如，要实现在网页中点击 p 标签来显示内容，之前我们会用下面的代码实现：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
</head>
<body>
    <p>我是第 1 个标签</p>
    <p>我是第 2 个标签</p>
    <p>我是第 3 个标签</p>
</body>
<script>
var p = document.getElementsByTagName('p');
for(var i =0;i<p.length;i++){
    p[i].onclick=function(){
    alert(this.innerHTML)
    }
}
</script>
</html>
```

使用 jQuery 选择器则可以这样实现，代码如下：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src = "https://code.jQuery.com/jQuery-2.1.1.js" ></script>
</head>
<body>
    <p>我是第 1 个标签</p>
    <p>我是第 2 个标签</p>
    <p>我是第 3 个标签</p>
</body>
<script>
var p = $('p');
p.click(function(){
```

```
        alert($(this).html());
    });
    </script>
    </html>
```

上面的两段代码运行效果是相同的，本例中使用$('p')获得了所有 p 标签的集合，并使用 html()函数显示 p 标签的内容。

此外，$还用来访问 jQuery 提供的额外方法及属性，比如$.map()、$.each()、$.ajax()。

$.map()将一组数组转换为其他数组，就是根据当前数组创建一个新的数组。它的语法为：

```
$.map(arr,callback);
```

它有两个参数，arr 为当前数组，callback 为执行的方法，用来改变这个数组。例如：

```
//声明一个数组
var arr = [1, 2, 3];
//将数组转换成另一个数组,第一个参数为要转换的数组，第二个参数为执行的方法
var arr2 = $.map(arr, function (item) { return item * 2; });
console.log(arr2);
//输出[2,4,6]
```

$.each()方法的语法是：

```
$.each(arrayOrObject,callback)
```

它的作用是遍历对象或数组。第一个参数为需要遍历的对象或数组，第二个参数是对每个数组成员执行的函数。例如：

```
var users = { "name": "王五", "age": 12 };
$.each(users, function (key, value) { console.log(key + "为: " + value) });
//输出 name 为：王五  age 为：12
```

还有其他一些实用函数，如$.inArray()、$.isEmptyObject()、$.isFunction()等。

jQuery 提供了一个叫做$(document).ready()的方法，它表示在 document“准备好”之后立即运行代码。重要的一点是，要运行的代码需要包含在一个匿名函数中，当 ready 事件触发时得到执行。它与 document.onload 的区别如表 6-1 所示。

表 6-1　window.onload 与$(document).ready()的区别

	window.onload	$(document).ready()
执行时机	等待网页中所有的资源加载完毕后(包括图片、flash、音频、视频)	DOM 树加载完毕后，不管里面的资源是否加载完成
编写个数	只能执行一个	可以执行多个
简化写法	无	$(function() {})

可见$(document).ready()是在 DOM 文档加载后就执行，而不必等待图片等文件载入，执行速度更快，而且在多个地方使用 ready()方法也不会发生冲突，比 window.onload 函数更具优越性。

6.1.4　jQuery 对象和 DOM 对象

jQuery 通过包装 DOM 对象后产生的新的对象(也就是 jQuery 对象)来访问并操作 DOM 元素。jQuery 对象提供了额外的方法，但是 jQuery 对象无法使用 DOM 对象的方法，DOM 对象也不能使用 jQuery 里的任何方法。

只需用$()把 DOM 对象包装起来就可以获得一个 jQuery 对象，例如：

```
var logo1 = $( "p");
```

但每次使用$()返回的对象都是不同的，即使是相同的选择器，例如：

```
var logo1 = $( "p" );
var logo2 = $( "p" );
alert(logo1 === logo2 ); // 输出 false
```

jQuery 对象其实是一个数组对象，因此可以通过数组的方法得到 DOM 对象。如果我们不得不使用 DOM 元素的方法，有如下两种方法将 jQuery 对象转换成 DOM 对象：

- 使用数组下标的方法，例如：$('p')[0]。
- 使用 jQuery 提供的 get()方法，该方法获得由选择器指定的 DOM 元素，它有一个可选的参数：index 编号，例如：$('p').get(0)表示获得第一个 p 元素。

在人人微博网站中个人主页的需求包括：

- 发布微博的字数不能超过 140 个字，当用户输入完成后自动提示还可以继续输入的字数，如果可输入的字数小于 25，则红色显示数字以提醒用户，如图 6-2 所示。

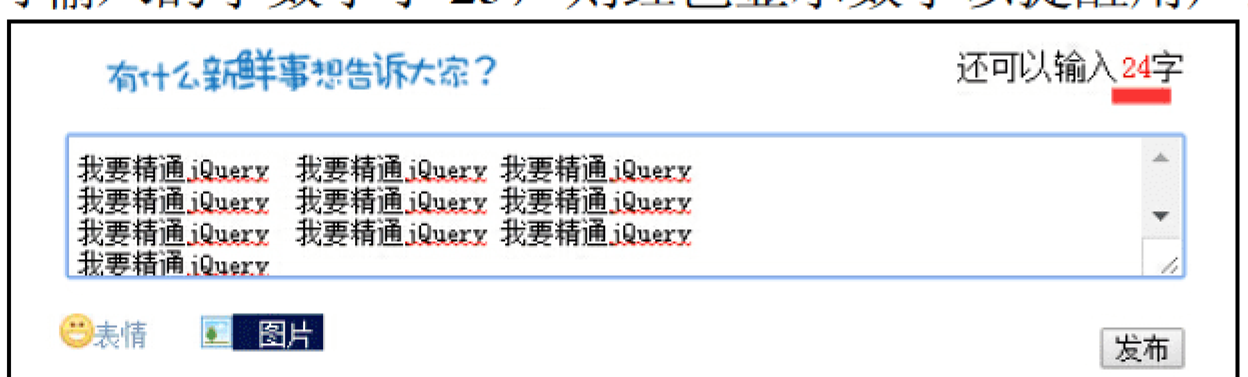


图 6-2　微博字数限制

- 微博中显示的图片是缩略图的形式，只有用户点击缩略图，才会显示大尺寸的图片。点击大图片后，重新显示缩略图，效果如图 6-3 所示。

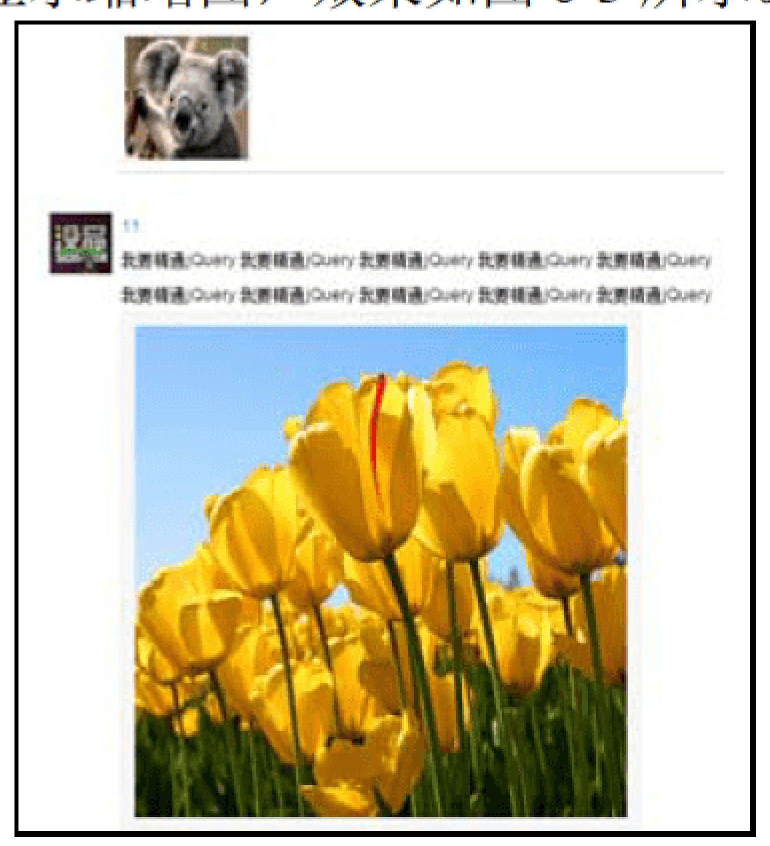


图 6-3　图片微博

本章将使用 jQuery 实现这两个功能。

6.2　jQuery 选择器

6.2.1　jQuery 基本选择器

jQuery 支持大多数 CSS 3.0 中的选择器，并自定义了一些选择器。基本选择器包括使用元素 id、class、元素名等，基本选择器可以实现大多数页面查找。详细说明如表 6-2 所示。

表 6-2　基本选择器

选 择 器	功　　能	返 回 值	示　　例
#id	根据给定的 ID 匹配一个元素	单个元素	$("#lastname")
element	根据给定的元素名匹配所有元素	元素集合	$("p")
.class	根据给定的 class 匹配元素	元素集合	$(".intro")
.class.class	根据给定的多个 class 匹配元素	元素集合	$(".intro.demo")
*	所有元素	元素集合	$("*")

对于下面的 HTML 代码示例，我们可以使用表 6-2 中的选择器来选择相应的元素：

```
<p>This is a paragraph, which means it is wrapped in <code>&lt;p&gt;</code>
    <span>and</span> <code>&lt;/p&gt;</code>. Those "p" tags in the previous
    sentence are formatted as <code>&lt;code&gt;</code>.</p>
<ul id="myUl">
  <li>This is the first list item (<code>&lt;li&gt;</code>) in an unordered
      list (<code>&lt;ul&gt;</code>). </li>
  <li>This is the second list item. It has a <a rel="self" title="Learning
      jQuery blog" href="/archives/jQuery-links.htm">link</a> in it.</li>
  <li class="myclass otherclass">This is the third list item. It has a
      <code>class</code> of "myclass otherclass"</li>
  <li>This is the fourth list item. It has <strong>strong</strong> text and
      <em>em</em>phasized <em>text</em>.
    <ul>
      <li>second-level list item 1</li>
      <li>second-level list item 2</li>
    </ul>
  </li>
</ul>
```

我们可以使用$('#myid')选择 ul 列表区域，使用$('code')选择所有的 code 元素的集合，使用$('.myclass')选择使用了 myclass 类的元素集合。还可以一次使用多个选择器，例如：

```
$(' code,.myclass ')
```

选择的区域如图 6-4 所示。

This is a paragraph, which means it is wrapped in <p> and </p>. Those "p" tags in the previous sentence are formatted as <code>.

- This is the first list item ( ) in an unordered list ( ).
- This is the second list item. It has a link in it.
- This is the third list item. It has a class of "myclass otherclass"
- This is the fourth list item. It has **strong** text and *emphasized text*.
  - second-level list item 1
  - second-level list item 2

图 6-4　选择的区域

6.2.2　jQuery 层次选择器

层次选择器通过 DOM 元素间的层次关系获取元素，其主要的层次关系包括后台、父子、相邻、兄弟关系，通过其中某类关系可以方便地定位元素，如表 6-3 所示。

表 6-3　层次选择器

选 择 器	功　　能	返 回 值	示　　例
ancestor descendant	根据祖先元素匹配所有的后代元素	元素集合	$("div span")
parent > child	根据父元素匹配所有的子元素	元素集合	$("#one > div")
prev + next	匹配所有紧接在 prev 元素后的相邻元素	元素集合	$("#one +div")
prev ~siblings	匹配 prev 元素之后的所有兄弟元素	元素集合	$("#one~div")

例如，使用$('strong + em')选择器的结果如图 6-5 所示。

- This is the fourth list item. It has **strong** text and *emphasized text*.

图 6-5　使用$('strong + em')选择器的结果

使用$('strong ~ em')选择器的结果如图 6-6 所示：

This is the fourth list item. It has **strong** text and *emphasized text*.

图 6-6　使用$('strong ~ em')选择器的结果

6.3　jQuery 过滤选择器

一般来说，通过选择器获得元素的集合后，就可以使用 jQuery 方法对其执行某种操作，比如用 hide()方法将其隐藏。这就需要从集合中筛选一部分元素，在这种情况下，可以在基本选择器的基础上添加过滤选择器来完成筛选任务。根据不同的具体情况，在过滤选择器中可以使用元素的索引值、内容、属性、子元素位置、表单域属性等作为筛选条件。下面来看看 jQuery 提供了哪些过滤器来帮助我们完成此任务。

6.3.1　jQuery 基本过滤

过滤选择器根据某类过滤规则进行元素的匹配，书写时都以冒号(:)开头，简单过滤选择

器是过滤选择器中使用最广泛的一种，其详细说明如表 6-4 所示。

表 6-4　基本过滤

选 择 器	功　　能	返 回 值
:first	获取第一个元素。等价于 first()函数	元素集合
:last	获取最后一个元素。等价于 last()函数	元素集合
:not(selector)	获取除给定选择器外的所有元素	元素集合
:even	匹配偶数号元素	元素集合
:odd	匹配奇数号元素	元素集合
:eq(n)	匹配页面中第 n 个元素。等价于 eq()函数	元素集合
:gt(n)	匹配页面中第 n 个元素之后的所有元素	元素集合
:lt(n)	匹配页面中第 n 个元素之前的所有元素	元素集合
:header	获取所有标题类型的元素	元素集合
:animated	获取正在执行动画效果的元素	元素集合
:hidden	获取所有不可见元素，或者 type 为 hidden 的元素	元素集合
:visible	获取所有可见元素	元素集合

6.3.2　jQuery 内容过滤

有时需要根据元素中的文字内容或所包含的子元素特征来匹配元素，这就是内容过滤选择器。其文字内容可以模糊或绝对匹配元素。详细说明如表 6-5 所示。

表 6-5　内容过滤

选 择 器	功　　能	返 回 值	示　　例
:contains(text)	选取包含内容为 text 的元素	元素集合	$("div:contains(' 我 ')") 选取含有“我”字的<div>元素
:empty	选取不包含子元素或文本的空元素	元素集合	$("div:empty")选取不包含子元素(包括文本)的<div>元素
:has(seletor)	选择含有选择器所匹配的元素的元素	元素集合	$("div:has(p)") 选取含有<p>元素的<div>元素
:parent	选择含有子元素或者文本的元素	元素集合	

6.3.3　jQuery 子元素过滤

有时需要根据子元素的位置进行匹配，比如，虽然使用基本过滤选择器:eq(index)可以实现单个表格的显示，但不能满足大量数据和多个表格的选择需求，比如表格子元素的个数动

态增加，如果匹配表格最后一个元素，显然固定的:eq(index)不能满足要求。这时可以使用子元素过滤选择器，详细说明如表 6-6 所示。

表 6-6　子元素过滤

选 择 器	功　　能	返 回 值
:nth-child(eq|even|odd|index)	获取每个父元素下的特定位置元素，索引号从 1 开始	元素集合
:first-child	获取每个父元素下的第一个子元素	元素集合
:last-child	获取每个父元素下的最后一个子元素	元素集合
:only-child	获取每个父元素下的仅一个子元素	元素集合

6.3.4　jQuery 属性过滤

元素中通常包含多个属性，除了使用 id 和 class 属性作为选择器外，还可以使用其他属性对集合进行过滤。进一步得到具有某个属性或属性满足一定规则的元素集合。属性过滤选择器包含在方括号内，而不是以冒号开头。详细说明如表 6-7 所示。

表 6-7　属性过滤

选 择 器	功　　能	返 回 值	示　　例
[attribute]	选取拥有此属性的元素	集合元素	$("div[id]")选取拥有属性 id 的元素
[attribute=value]	选取属性的值为 value 的元素	集合元素	$("div[title=test]") 选取属性 title 为"test"的<div>元素
[attribute!=value]	选取属性的值不等于 value 的元素	集合元素	$("div[title!=test]") 选取属性 title 不等于"test"的<div>元素(注：没有属性 title 的<div>元素也会被选取)
[attribute^=value]	选取属性的值以 value 开始的元素	集合元素	$("div[title^=test]") 选取属性 title 以"test"开头的<div>元素
[attribute$=value]	选取属性的值以 value 结尾的元素	集合元素	$("div[title$=test]") 选取属性 title 以"test"结束的<div>元素
[attribute*=value]	选取属性的值含有 value 的元素	集合元素	$("div[title*=test]") 选取属性 title 含有"test"的<div>元素
[selector1][selector2][selector]	用属性选择器合并成一个复合的属性选择器，满足多个条件。每选取一次，缩小一次范围	集合元素	$("div[id][title$='test']")选取拥有属性 id，并且属性 title 以"test"结束的<div>元素

6.3.5　jQuery 表单属性过滤

为了提高使用表单的效率，在 jQuery 选择器中引入表单属性过滤选择器，该选择器专为表单量身打造，通过它可以在页面中快速定位某表单对象。既可以通过表单中元素的类型，也可以通过表单中元素的当前状态进行过滤。详细说明如表 6-8 所示。

表 6-8　表单属性过滤

选 择 器	功　　能	返 回 值	示　　例
:input	选取所有的 input、textarea、button、select 元素	集合元素	
:text	选取所有的单行文本框	集合元素	$(":text")选取所有的单行文本框
:password	选取所有的密码框	集合元素	$(":password")选取所有的密码框
:radio	选取所有的单选框	集合元素	$(":radio")选取所有的单选框
:checkbox	选取所有的多选框	集合元素	$(":checkbox")选取所有的多选框
:submit	选取所有的提交按钮	集合元素	$(":submit")选取所有的提交按钮
:image	选取所有的图形	集合元素	$(":image")选取所有的图形
:reset	选取所有的重置按钮	集合元素	$(":reset")选取所有的重置按钮
:button	选取所有的按钮	集合元素	$(":button")选取所有的按钮
:file	选取所有的上传域	集合元素	$(":file") 选取所有的上传域
:enabled	选取所有可用元素	集合元素	$("#form1 :enabled")选取 id 为"form1"的表单中的所有可用元素
:disabled	选取所有不可用元素	集合元素	$("#form1 :disabled")选取 id 为"form1"的表单中的所有不可用元素
:checked	选取所有被选中的元素(单选框、复选框)	集合元素	$("input :checked")选取所有被选中的<input>元素
:selected	选取所有被选中的选项元素(下拉列表)	集合元素	$("select :selected")选取所有被选中的选项元素

6.4　jQuery 操作 DOM 元素

6.4.1　元素的创建

我们已经熟悉 DOM 模型，它由页面中的各个元素形成的节点相互关联形成树状结构。jQuery 可以动态地创建 DOM 元素，并添加到指定的节点位置。动态创建页面元素的语法如下：

```
$(html)
```

html 表示有效的 html 标记字符串，例如：

```
var article=
$(<article id='weibo1' class='weibo'></article>)
```

或者使用下面的代码：

```
$( "<article/>", {id:"weibo1",class:"weibo"})
```

6.4.2　元素的插入

动态地创建完成 DOM 元素后，还需要执行节点的插入或追加操作。表 6-9 中的 4 个方法可以很容易地添加新元素。

表 6-9　元素的插入

方　法	描　述
append()	向每个匹配元素的内部的结尾处追加内容
appendTo()	将内容追加到指定元素的内部的结尾处
prepend()	向每个匹配元素的内部的开始处插入内容
prependTo()	将内容插入到指定元素的内部的开始处
after()	向每个匹配的元素之后插入内容
insertAfter()	将每个匹配的元素插入到指定元素之后
before()	向每个匹配的元素之前插入内容
insertBefore()	将每个匹配的元素插入到指定元素之前

6.4.3　元素的删除

元素的删除分为删除指定的 DOM 元素和从指定元素中删除所有的子节点。jQuery 提供了 3 种方法：remove()方法、empty()方法和 detach()方法。

remove()方法会删除 DOM 元素本身和所有的子元素。例如：

```
<p>Hello</p>World<p>jQuery</p>
```

执行$("p").remove()语句，得到的结果是 world。

empty()方法只是从 DOM 元素中删除子元素，同样是上面的 html 代码，如果执行$("p").empty()语句，得到的结果是<p></p>World<p></p>。

detach()方法与 remove()方法基本相同，也是从 DOM 中删除所有匹配的元素，但与 remove()方法不同的是，被删除元素的所有绑定的事件、附加的数据等都会保留下来。因此，如果以后要将移除的元素重新插入到 DOM 中，detach()方法将非常有用。

上述 3 个方法的区别如表 6-10 所示。

表 6-10　删除元素的方法的区别

方　法	元素本身(包含所有属性)	绑定事件和附加的相关数据	文本域及子节点
empty()	不删除	不删除	删除
remove()	删除	删除	删除
detach()	删除	不删除	删除

6.4.4　元素的包裹

所谓元素包裹是指用另外一个指定的标记将匹配的元素包裹起来，并返回一个 jQuery 对象。

1. wrap()方法

wrap()方法的语法是：

```
$(selector).wrap( element | function)
```

它的参数可以是一个选择器、DOM 元素、html 代码、jQuery 对象、函数。例如：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>wrap demo</title>
  <style>
  div {
    border: 2px solid blue;
  }
  p {
    background: yellow;
    margin: 4px;
  }
  </style>
  <script src="https://code.jQuery.com/jQuery-1.10.2.js"></script>
</head>
<body>
<p>Hello</p>
<p>jQuery</p>
<p>World</p>
<script>
$( "p" ).wrap( "<div></div>" );
</script>
</body>
</html>
```

上面的代码对每一个 p 元素用 div 元素包裹起来。显示效果如图 6-7 所示。

Hello

jquery

World

图 6-7　wrap()方法的显示效果

如果参数是一个回调函数，它将生成一个用来包裹匹配元素的 HTML 结构。例如：

```
<div class="container">
  <div class="inner">Hello</div>
  <div class="inner">Goodbye</div>
</div>
<script>
$( ".inner" ).wrap(function() {
  return "<div class='" + $( this ).text() + "'></div>";
});
</script>
```

会得到如下的 HTML 结构：

```
<div class="container">
  <div class="Hello">
    <div class="inner">Hello</div>
  </div>
  <div class="Goodbye">
    <div class="inner">Goodbye</div>
  </div>
</div>
```

与 wrap()方法作用相反的方法是 unwrap()，它从 DOM 中删除匹配元素中每个元素的父元素(直接上级元素)。例如：

```
<div>
<p>Hello</p>
</div>
<div>
<p>jQuery</p>
</div>
<div>
<p>World</p>
</div>
<script>
$( "p" ).unwrap();
</script>
```

将会得到下面的 HTML 结构：

```
<p>Hello</p>
<p>jQuery</p>
<p>World</p>
```

2. wrapInner()方法

此方法使用一个 HTML 结构来包装匹配集合中每个元素的内容(包括文本节点在内)，并返回一个 jQuery 对象。例如，对如下的 HTML 代码：

```
<div id ="div1">
<p>Hello</p>
<p>World</p>
</div>
```

执行如下 jQuery 语句：

```
$("#div1 p").wrapInner("<b class = 'bold' />");
```

得到的结果是：

```
<div id ="div1">
<p><b class  = 'bold' >Hello</b></p>
<p><b class  = 'bold' >World</b></p>
</div>
```

3. wrapAll 方法

此方法使用一个 HTML 结构元素包裹在所有匹配元素的周围，并返回 jQuery 对象。例如：

```
<div id ="div1">
<p>Hello</p>
<p>World</p>
</div>
```

执行如下的 jQuery 语句：

```
$("#div1 p").wrapAll("<div class = 'inner'/>");
```

得到的结果是：

```
<div id ="div1">
<div class = 'inner'>
<p>Hello</p>
<p>World</p>
</div>
</div>
```

6.4.5　元素的替换和复制

jQuery 提供了替换和复制 DOM 元素的方法，也就是将原来的元素删除后再插入一个新元素。

1. replaceWith()方法

```
<html>
<head>
<meta charset="utf-8">
<script src="https://code.jQuery.com/jQuery-2.1.1.js"></script>
<script type="text/javascript">
$(document).ready(function(){
  $(".btn1").click(function(){
    $("p").replaceWith("<b>Hello world!</b>");
  });
});
</script>
</head>
<body>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button class="btn1">用粗体文本替换所有段落</button>
</body>
</html>
```

点击按钮后的效果是使用粗体的“Hello world!”替换了原来的 p 元素。

2. replaceAll()方法

此方法使用匹配元素集合来替换每个目标对象，并返回 jQuery 对象。它的参数可以是 DOM 元素、jQuery 对象、选择器、数组。例如：

```
$("<div>").replaceAll("p");
```

会将所有 p 元素及其内容替换为 div 元素

3. clone()方法

使用 clone()方法可执行复制元素的操作，以创建匹配元素集合的副本。由于复制元素后需要将新元素再插入文档中，因此 clone()方法通常和某种插入方法结合使用。它的语法为：

```
$(selector).clone( [withDataAndEvents ] [, deepWithDataAndEvents ] )
```

withDataAndEvents 是布尔值，表示同时拷贝元素的数据和绑定的事件。默认为 false。

deepWithDataAndEvents 是布尔值，表示同时拷贝元素及其子元素的数据和绑定的事件，默认为 false。

例如：

```
<div class="container">
  <div class="hello">Hello</div>
  <div class="goodbye">Goodbye</div>
</div>
<script>
$( ".hello" ).clone().appendTo( ".goodbye" );
</script>
```

执行后，会得到如下 HTML 代码：

```
<div class="container">
  <div class="hello">Hello</div>
  <div class="goodbye">
    Goodbye
    <div class="hello">Hello</div>
  </div>
</div>
```

6.4.6　遍历和筛选 DOM 元素

1. .is()

根据选择器检查当前匹配元素集合，如果存在至少一个匹配元素，则返回 true，它对结果集求值但并不改动结果集。一般用在回调函数(callback)中。例如：

```
$("p").is(".foo");
```

这行代码表示先选出所有的段落，再检查是否有一个段落拥有 class foo。

2. .find()

用来在当前结果集中查找符合条件的后代元素，它的查找范围不限制是子元素还是多少代孙元素。例如：

```
<body>
  <p><span>Hello</span>, how are you?</p>
  <p>Me? I'm <span>good</span>.</p>
 <span>jQuery</span>
<script>
  $("body").find("span").css('color','red');
</script>
</body>
```

上面的代码表示搜索所有 body 中的后代 span 元素，并将其颜色设置为红色，显示结果为 Hello、good、jQuery 成为红色字体。

3. .not()

从匹配元素集合中删除元素，唯一的参数可以是选择器或者 DOM 元素。例如：

```
$("p").not("#selected")
```

从包含所有段落的集合中删除 id 为"selected"的段落。

4. .addSelf()

它可以让我们在使用一个遍历方法查找目标元素时，保留原始结果集。例如：

```
$("p").find("span");
```

只包含查找到的 span 子元素，如果想把原始结果集(p 元素集合)合并到查找结果当中，就应该使用下面的代码：

```
$("p").find("span").andSelf();
```

5. .children()

获得匹配元素集合中每个元素的所有子元素。

例如：

```
$("p").children(".foo");
```

这行代码表示获取所有段落中拥有 class foo 的子元素。

6. .parent()

获得当前匹配元素集合中每个元素的直接父元素。例如：

```
$("span ").parent();
```

上面的代码表示选出页面 span 元素的直接父元素。

7. .closest()

.closest()方法允许我们检索 DOM 树中的这些元素以及它们的祖先元素，并用匹配元素构造新的 jQuery 对象。

与.parents()的区别如表 6-11 所示。

表 6-11　.closest()与.parents()方法的区别

.closest()	.parents()
从当前元素开始	从父元素开始
沿 DOM 树向上遍历，直到找到已应用选择器的第一个匹配为止	沿 DOM 树向上遍历，直到文档的根元素为止，将每个祖先元素添加到一个临时的集合；如果应用了选择器，则会基于该选择器对这个集合进行筛选
返回包含零个或一个元素的 jQuery 对象	返回包含零个、一个或多个元素的 jQuery 对象

8. .parents()

与.parent()不同，.parents()会返回所有的祖先元素，它接受一个可选的参数来过滤结果集。例如：

```
$("span ").parents();
```

上面的代码表示选出页面 span 元素的全部祖先元素。

9. .next()和.nextAll()

.next()方法将找出原始结果集中每一个元素的下一个兄弟元素。而.nextAll()方法将找出原始结果集中每一个元素之后的所有兄弟元素。

10. .prev()和.prevAll()

.prev()、.prevAll()方法与.next()、.nextAll()方法只有一点不同，那就是它们是向前而不是向后查找兄弟元素。

11. .siblings()

获得匹配元素集合中所有元素的同辈元素，由选择器筛选(可选)。它会选择全部兄弟元素(不分前后的同辈元素)。

6.5　jQuery 对 DOM 属性操作

jQuery 能够通过指定 DOM 元素的属性给网页带来活力，比如指定 src、class、title、width 等属性。这在客户端脚本编程中十分重要。

6.5.1　获取和设置元素属性

jQuery 使用 attr()方法读取和设置元素的属性，当为该方法传递一个参数时，即为获取某元素的指定属性；当为该方法传递两个参数时，即为设置元素指定属性的值。例如：

```
//读取
var src = $("img").attr("src");
//设置
//$("img").attr("src","logo.jpg");
```

设置属性时第二个参数可以是函数，当新设置的属性值依赖于属性原来的值或其他值时，就要使用到函数的强大之处。例如：

```
$("p").attr('title',function(index,previousValue) {
    return previousValue + '我的编号'+index;
});
```

在上面的代码中，index 参数是元素在集合中从零开始的下标，previousValue 是元素属性的当前值。上面的代码会作用于所有的 p 元素，并为每个 p 元素的 title 属性指定不同的值。

如果需要一次设置多个属性的值，传入一个包含属性键值对的对象即可，例如：

```
$('input').attr({value:'',title:'请输入'});
```

removeAttr() 方法从被选元素中移除属性。例如：

```
$("img").removeAttr ("src")
```

此外，val()方法也可以返回或设置元素的 value 属性值，该方法大多用于 input 元素。例如：

```
//读取
$(":email").val();
//设置
$(":email").val("example@163.com");
```

6.5.2　获取和设置元素内容

jQuery 可以动态地获取和设置元素的内容，并提供 html()方法来获取和设置元素的 HTML 内容，提供 text()方法来获取和设置元素的文本内容。

1. html()方法

不提供参数时会获取匹配元素的集合中第一个元素的 HTML 内容并返回字符串。如果提供了字符串参数，则根据参数来设置集合中每个元素的 HTML 内容并返回 jQuery 对象。例如：

```
<body>
<div>
</div>
 <div>
</div>
<script>
$("div").html("<p>Nice to meet you</p>");
alert($("div").html());
//输出<p>Nice to meet you</p>
</script>
</body>
```

上面的代码的运行效果如图 6-8 所示。

```
Nice to meet you

Nice to meet you
```

图 6-8　html()方法的运行效果

2. text()方法

不提供参数时会获取匹配元素集合中所有元素的文本内容，而不像 html()方法只获取第一个元素的 HTML 内容。如果提供了字符串参数，则会根据参数来设置集合中每个元素的文本内容，并返回 jQuery 对象。例如：

```
<body>
<div>
</div>
 <div>
</div>
<script>
$("div").text("<p>Nice to meet you</p>");
```

```
alert($("div").text());
//输出<p>Nice to meet you</p> <p>Nice to meet you</p>
</script>
</body>
```

上面的代码的运行效果如图 6-9 所示。

```
<p>Nice to meet you</p>
<p>Nice to meet you</p>
```

图 6-9　text()方法的运行效果

6.5.3　获取和设置元素的 CSS 属性

1. css()方法

jQuery 提供了 css()方法来获取和设置元素的 CSS 样式属性，其方法类似于 attr()方法。可以通过指定名称和值来设置单个的 CSS 样式，也可以通过传入一个对象来设置一系列的 CSS 样式。例如：

```
<body>
 <h1>前端开发</h1>
 <p>jQuery</p>
<script>
 $("h1").css("border","10px solid")
 $("p").css({
   "textDecoration":"underline",
   "background-color":"#b7b7ff"
 });
 </script>
</body>
```

上面的代码的运行效果如图 6-10 所示。

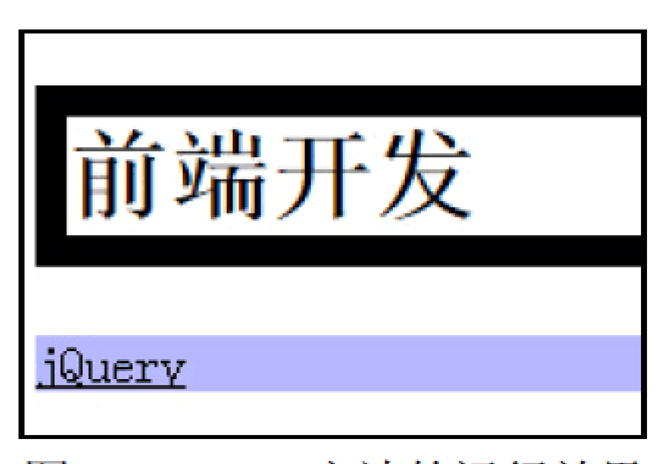


图 6-10　css()方法的运行效果

需要注意的是，CSS 属性应该写为 JavaScript 中的形式，如 text-decoration 写成 textDecoration。

2. addClass()和 removeClass()方法

为了将样式信息保留在 CSS 文件中，jQuery 提供了添加和移除 CSS 类的函数。addClass()向一个或一组元素中添加一个或多个类，removeClass()方法从一个或一组元素中移除一个或多个类。这两个方法接受一个字符串参数，表示即将作用于目标元素的类，如果是由空格分隔的字符串，则表示应用多个类。返回的值是更改样式后的元素集合。例如：

```
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("p:first").addClass("intro");
  });
});
</script>
<style type="text/css">
.intro
{
font-size:220%;
background-color:red;
}
</style>
</head>

<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>向第一个 p 元素添加一个类</button>
</body>
```

这段代码展示了 addClass()方法的实际作用，当点击按钮后，第一个 p 标签的字体大小和颜色发生变化。

3. hasClass()和 toggleClass()方法

判断元素是否包含特定的类是很常见的需求。有时需要根据元素是否包含特定的类来进行条件判断。这时就可以使用 hasClass()方法来实现，它的语法为：

```
hasClass(name)
```

name 表示要检查的类名，为字符串格式。如果集合中包含传入参数的类名，则返回 true，否则返回 false。

还有一种普遍的操作是根据元素的类在两个状态之间变化，也就是进行类的切换。jQuery 提供了 toggleClass()方法简化操作，它的语法为：

```
toggleClass(name)
```

name 表示要切换的单个类名，或者以空格分隔的字符串表示多个类名。如果元素不存在指定类名，则为其添加类名；如果元素已经拥有这个类名，则从中删除此类名。返回的是元素集合。例如：

```
<table>
     <tr>
       <th>ID</th>
       <th>Fruit</th>
       <th>Price</th>
```

```
        </tr>
        <tr>
          <td>1</td>
          <td>Apple</td>
          <td>0.60</td>
        </tr>
        <tr>
          <td>2</td>
          <td>Orange</td>
          <td>0.50</td>
        </tr>
        <tr>
          <td>3</td>
          <td>Banana</td>
          <td>0.10</td>
        </tr>
    </table>
    <script type="text/javascript">
        $(function() {
          $("table tr:nth-child(even)").addClass("striped");
          function swap() {
            $( this ).toggleClass('striped');
          }
          $("tr").mouseover(swap).mouseout(swap);
        });
      </script>
```

在价格表中，我们希望表格中的行交替设置不同的颜色，当鼠标移入移出的时候，将奇数行的彩色背景与偶数行的背景进行互换。在 JavaScript 代码中，我们使用了学习过的 nth-child 选择器，为表格中的隔行元素应用类 striped，定义了为所有 tr 元素切换背景颜色的函数 swap。最后是重点，使用 mouseover 和 mouseout 事件处理函数，分别调用 toggleClass() 方法。得到的结果是每当鼠标进入或者移出行时，就会在当前行应用 stripped 类或者删除 stripped 类。

6.6　jQuery 响应事件

我们已经学习过 ready()事件处理函数，事件在触发后被分为两个阶段，一个是捕获阶段，另一个是冒泡阶段。jQuery 定义了兼容性更好的事件名称，更加简洁的事件处理方式。包括鼠标事件、表单事件、浏览器事件和文档事件等。jQuery 的事件名与标准 DOM 中的事件名很相似。

表 6-12 给出了 jQuery 中的鼠标键盘事件。

表 6-12　jQuery 中的鼠标键盘事件

事 件 名	说　　明
click	当鼠标指针位于一个元素上时，如果单击鼠标按钮，则会触发该元素的 click 事件
dbclick	当双击一个元素时，就会向该元素发送 dbclick 事件
hover()	当鼠标移入到一个元素上面及移出这个元素时触发
mousedown	按下鼠标按键时触发
mouseup	释放鼠标按键时触发
mousemove	当鼠标在某个元素内移动时触发
mouseenter	当鼠标进入某个元素时触发
mouseleave	当鼠标离开某个元素时触发
mouseover	当鼠标指针进入某个元素时触发
mouseout	当鼠标指针离开某个元素时触发
keydown	在键盘按下时触发
keyup	在按键释放时触发
keypress	在敲击按键时触发

用 jQuery 处理事件的语法是：

```
$(document).ready(function() {
    $("#btn").click(function(){
        alert("jQuery");
    });
})
```

jQuery 提供的 mouseenter 和 mouseleave 事件与 mouseover 和 mouseout 事件比较相似，但 mouseenter 和 mouseleave 事件不存在冒泡现象。由于 mouseover 和 mouseout 事件存在冒泡现象，当一个元素内还有子元素时，鼠标进入该元素的子元素的边界之内时又会触发 mouseover 事件，并将事件冒泡传递给外层的该元素。

hover()方法实际上等价于 mouseenter 与 mouseleave 事件的组合。例如：

```
<ul>
  <li>Milk</li>
  <li>Bread</li>
</ul>
<script>
$( "li" ).hover(
  function() {
    $( this ).append( $( "<span> ***</span>" ) );
  }, function() {
    $( this ).find( "span" ).remove();
  }
);
</script>
```

表 6-13 给出了 jQuery 中的表单事件。

表 6-13　jQuery 中的表单事件

事 件 名	说　明
blur	当一个元素失去焦点时触发
focus	当一个元素获得焦点时触发
change	当文本框、文本区域和选择框的值发生变化时触发
focusin	当元素或子元素获得焦点时触发
focusout	当元素或子元素失去焦点时触发
select	当 textarea 或文本类型的 input 元素中的文本被选择时会发生 select 事件
submit	用户提交表单时触发

focusout 和 blur 的区别是 focusout 事件存在冒泡现象，focusin 和 focus 的区别也是如此。当一个元素内还有子元素时，子元素失去焦点也会触发父元素上的 focusout 事件。例如：

```
<div class="inputs">
  <p>
    <input type="text"><br>
    <input type="text">
  </p>
  <p>
    <input type="password">
  </p>
</div>
<div id="focus-count">focusout fire</div>
<div id="blur-count">blur fire</div>

<script>
var focus = 0,
  blur = 0;
$( "p" )
  .focusout(function() {
    focus++;
    $( "#focus-count" ).text( "focusout fired: " + focus + "x" );
  })
  .blur(function() {
    blur++;
    $( "#blur-count" ).text( "blur fired: " + blur + "x" );
  });
</script>
```

运行上面的代码时，p 元素内的 input 元素失去焦点时，会触发 p 元素上的 focusout 处理函数而不会触发 blur 处理函数。

下面我们来实现点击缩略图时显示大图特效，需要实现的功能包括：①点击缩略图后，出现 loading(正在加载)的小图标，然后加载大图，当大图加载完成后隐藏 loading 的小图标并

显示大图；②点击大图后，隐藏大图并显示缩略图。代码实现如下：

HTML5 代码为：

```
<div class="cont_down41">
  <!--用户头像-->
  <div class="cont_down4_a"><img class="picture1" /></div>
  <div class="cont_down4_b">
    <!--用户昵称-->
    <div class="fo" ></div>
    <!--微博内容-->
    <div data-content="content"></div>
    <!--缩略图 div-->
    <div class="photoBox">
     <div class="loadingBox"> <span class="loading"></span> </div>
     <img src="images/imgtest_small.jpg" class="zoom" onclick=
          "zoom_image($(this).parent());"/> </div>
    <!--大图 div-->
    <div class="photoArea" style="display:none;"> <img src="" class=
         "minifier" onclick="zoom_image($(this).parent());" /> </div>
    <div>
     <!--微博发布时间称-->
     <time data-content="timestamp" data-format="DateTimeFormatter"> </time>
    </div>
  </div>
</div>
```

样式代码为：

```
.photoBox{width:100px;height:100px;background:#fff;padding:2px;border:1px
    solid #E5E5E5;display:inline-block;margin:10px 20px 0 0;position:relative;}
.photoBox img{width:100px;   height:100px;}
.loadingBox{background:#fff;*filter:alpha(opacity=30);text-align:center;
    margin:-8px 0 0 -8px;display:none;}
.loading{background:url(../images/loading_16.gif) no-repeat 0 0;
    width:16px;height:16px;display:inline-block;}
.photoArea{width:400px;  height:400px;background:#F7F7F7;
    border:1px solid #EBEBEB;padding:10px;text-align:center;zoom:1;}
```

jQuery 代码为：

```
<script>
function zoom_image(obj) {
    if (obj.hasClass('photoBox')) {
        var load = obj.find('.loadingBox');
        load.show();
        var img = obj.next().find('img');
        if (img.attr('src') === '') {
          img.attr('src', obj.find('img').attr('src').replace('_small.', '.'));
            img.load(function() {
                obj.hide();
                obj.next().show();
```

```
            });
        } else {
            obj.hide();
            obj.next().show();
        }
    } else {
        obj.hide();
        obj.prev().show();
        obj.prev().find('.loadingBox').hide();
    }
</script>
```

表 6-14 给出了 jQuery 中的浏览器事件。

表 6-14　jQuery 中的浏览器事件

事 件 名	说　　明
.error()	未正确加载文档中某个元素时触发
.resize()	当对象的大小将要发生变化时触发
.scroll()	要滚动元素时触发

表 6-15 给出了 jQuery 中的文档事件。

表 6-15　jQuery 中的文档事件

事 件 名	说　　明
.load()	当指定的元素(及子元素)已加载时，会发生 load()事件。该事件适用于任何带有 URL 的元素(比如图像、脚本)
.ready()	当 DOM 树加载完毕后触发
.unload()	离开页面时触发(浏览器窗口已被关闭、使用前进或后退按钮)

6.6.1　绑定事件

将事件处理函数绑定到元素的事件上的方法是 on()方法，语法如下：

```
$(selector).on(event,childSelector,data,function)
```

event：表示由空格分隔多个事件名称。必须是有效的事件。

childSelector：可选，表示只能添加到指定的子元素上的事件处理程序。

data：可选。规定传递到函数的额外数据。

function：规定当事件发生时运行的函数。

例如：

```
function greet( event ) {
  alert( "Hello " + event.data.name );
}
$( "button" ).on( "click", {
```

```
  name: "Karl"
}, greet );
```

在人人微博网站中，如果发布的微博中有指向网站外部的链接，我们希望在新的浏览器窗口打开。这里约定只有外部的链接才使用 http 开头。实现代码如下：

```
$( "#someDiv" ).on( "click", "a", function( event ) {
    var elem = $( this );
    if ( elem.is( "[href^='http']" ) ) {
        elem.attr( "target", "_blank" );
    }
});
```

one()方法为每一个匹配元素绑定一个一次性的事件处理函数。也就是事件处理函数只会被执行一次。例如：

```
$( "p" ).one( "click", function() {
  alert( $( this ).text() );
});
```

自 jQuery 1.7 以来，on()函数的功能得到增强，在此之前绑定事件的 bind()、delegate()和 live()已经不推荐使用，对事件的绑定最好使用 on()函数。

6.6.2　移除事件

off()方法通常用于移除通过 on() 方法添加的事件处理程序。它的语法为：

```
.off( events [, selector ] [, handler ] )
```

例如：

```
$("button").click(function(){
        $("p").off("click");
});
```

6.6.3　Event 实例

jQuery 的 Event 实例还拥有额外的两个方法，如表 6-16 所示。

表 6-16　Event 方法

方　　法	说　　明
stopImmediatePropagation()	取消执行其他的事件处理函数并取消事件冒泡。如果同一个事件绑定了多个事件处理函数，则在其中一个事件处理函数中调用此方法后将不会继续调用其他的事件处理函数
isImmediatePropagationStopped()	是否调用过 stopImmediatePropagation()方法

6.6.4　触发事件

jQuery 提供了 trigger 方法来触发事件，即使用户不操作界面也能够触发各种事件。比如，如果用 trigger()触发一个 submit 事件，则同样会导致浏览器提交表单。所有触发的事件现在会冒泡到 DOM 树上。我们可以用 stopPropagation() 来阻止事件冒泡，或者在事件处理函数中返回 false 即可。语法为：

```
.trigger( eventType [, extraParameters ] )
```

eventType 参数表示事件类型，可以是有效的字符串或者 jQuery Event 对象。

extraParameters 参数表示传递到事件处理程序的额外参数。

例如：

```
$( "#foo" ).on( "click", function(event, a, b) {
  alert( $( this ).text() );
});
$( "#foo" ).trigger( "click",[ "foo", "bar" ] );

var event = jQuery.Event( "submit" );
$( "form:first" ).trigger( event );
if ( event.isDefaultPrevented() ) {
  // Perform an action...
}
```

6.6.5　自定义事件

所谓自定义事件，就是有别于带有浏览器特定行为的事件(类似 click、mouseover、submit、keydown 等事件)。例如：

```
$("p").on("myEvent", function (event, message1, message2) {
       alert(message1 + ' ' + message2);
});
$("p").trigger("myEvent", ["Hello","World!"]);
```

jQuery 的自定义事件还有命名空间机制，jQuery 的事件命名空间机制提供了在同一个事件下对处理函数再分类的功能。例如：

```
$(document).on("click.c1",function(){console.log("c1");});
$(document).on("click.c2",function(){console.log("c2");});
```

如果想移除 click.c1 事件处理函数，保留 click.c2 处理函数，使用$(document).off("click")是行不通的，因为$(document).off("click")会移除 click.c1 和 click.c2 处理函数。这时候就可以使用命名空间，使用$(document).off("click.c1")只会移除 click.c1 处理函数。

那我们为什么使用自定义事件呢？例如厨房中有一个灯泡和两个开关，两个开关都可以控制灯泡的状态。卧室中也有一个灯泡和两个开关，此外还有一个总开关，如果有任何灯泡处于开启状态，切换总开关会关闭所有灯泡，否则切换总开关会打开所有灯泡。如果不使用

自定义事件和命名空间，实现代码如下所示：

```
<div class="room" id="kitchen">
   <div class="lightbulb on">灯泡</div>
   <div class="switch">开关</div>
   <div class="switch">开关</div>
</div>
<div class="room" id="bedroom">
   <div class="lightbulb on">灯泡</div>
   <div class="switch">开关</div>
   <div class="switch">开关</div>
</div>
<div id="master_switch">总开关</div>
</div>
<script type="text/javascript">
$( ".switch" ).click(function() {
   var light = $( this ).closest( ".room" ).find( ".lightbulb" );
   if ( light.is( ".on" ) ) {
      light.removeClass( "on" ).addClass( "off" );
   } else {
      light.removeClass( "off" ).addClass( "on" );
   }
});
$( "#master_switch" ).click(function() {
   var lightbulbs = $( ".lightbulb" );
   //检查是否有任何灯泡处于开启状态
   if ( lightbulbs.is( ".on" ) ) {
      lightbulbs.removeClass( "on" ).addClass( "off" );
   } else {
      lightbulbs.removeClass( "off" ).addClass( "on" );
   }
});
</script>
```

如果我们使用自定义事件，并分析后添加 light:on(开灯)、light:off(关灯)、light:toggle(切换灯泡状态)三种自定义事件，则实现代码如下：

```
<script type="text/javascript">
//在灯泡上绑定三种事件
$( ".lightbulb" ).on( "light:toggle", function( event ) {
   var light = $( this );
   if ( light.is( ".on" ) ) {
      light.trigger( "light:off" );
   } else {
      light.trigger( "light:on" );
   }
}).on( "light:on", function( event ) {
   $( this ).removeClass( "off" ).addClass( "on" );
}).on( "light:off", function( event ) {
   $( this ).removeClass( "on" ).addClass( "off" );
});
```

```
//开关点击事件
$( ".switch" ).click(function() {
    var room = $( this ).closest( ".room" );
    room.find( ".lightbulb" ).trigger( "light:toggle" );
});
$( "#master_switch" ).click(function() {
    var lightbulbs = $( ".lightbulb" );
    // 检查是否有任何灯泡处于开启状态
    if ( lightbulbs.is( ".on" ) ) {
        lightbulbs.trigger( "light:off" );
    } else {
        lightbulbs.trigger( "light:on" );
    }
});
</script>
```

这段代码看起来复杂了，但是代码结构变得更好了，行为处理代码与目标元素绑定在一起，这样更易读和易于维护。自定义事件开启了前端基于事件驱动编程的大门。

下面我们来实现输入微博时字数限制的功能。

HTML5 代码如下：

```
<div id="word"><img src="images/send_weibo.png" />
    <div  class="aviableCount" >还可以输入<span>140</span>字
    </div></div>
    <div id="box">
        <div class="box1">
        //设置最大输入字符长度为 140
        <textarea name="weiboTextArea" id = "weiboTextArea"   class="box2"
            maxlength="140" ></textarea>
        </div>
</div>
```

样式代码如下：

```
#word{
    margin:15px 0px 0px 20px;
    padding:0px;
}
#box{
    //background-color:#063;
    width:550px;
    height:90px;
}
.box1{
    width:542px;
    height:50px;
    margin:7px 0px 0px 0px;
    padding:2px 3px 0px 3px;
}
.aviableCount{
    float: right;
}
```

```
.warning{
    color: red;
}
```

jQuery 代码如下：

```
//全局变量
var defaults = {
     allowed: 140,
     warning: 25,
     cssWarning: 'warning',
     };
//添加事件
$("#weiboTextArea").on('keyup', function(event) {
     event.preventDefault();
     var count = $(this).val().length;
     var available = defaults.allowed - count;
     if(available <= defaults.warning && available >= 0){
      //设置可输入字数的样式
       $(".aviableCount span").addClass(defaults.cssWarning);
     } else {
       $(".aviableCount span").removeClass(defaults.cssWarning);
     }
    //动态更新可以输入的字数
     $(".aviableCount span").html(available);
 });
```

本章小结

本章我们学习了 jQuery 的页面操作的各种方法和事件处理机制。利用目前所学的知识，我们可以使用强大的选择器来选择元素，然后将它们移动到页面的任意部位，也可以复制、移动、替换元素，甚至从头创建全新的元素。而实现这些功能的代码都非常简洁。

最后我们学习了 jQuery 事件、触发事件和自定义事件。我们了解了在处理 Web 页面事件时一些棘手的挑战，这些技能对于创建交互式的 Web 页面是必不可少的。

本章练习

1. jQuery 的元素选择器有哪几种？
2. jQuery 中绑定事件和移除事件的方法是什么？
3. 谈一谈对事件驱动编程的理解。

第7章　基于jQuery的Ajax技术

在 2005 年，Jesse James Garrett 创建了术语 Ajax(Asynchronous JavaScript and XML)，它是异步 JavaScript 和 XML 的英文缩写。这项技术以新的方式改变了 Web 的外观，不必加载整个页面就可以向服务器发起异步请求，用户可以在页面无刷新的情况下与服务器交互。

其实早在 1998 年，微软就引入了一个 ActiveX 控件，从而能够在脚本控制下执行异步请求，但当时似乎很少有人关注其中潜在的技术。又过了几年，几个非微软浏览器将该技术的标准化版本实现为 XMLHttpRequest(XHR)对象。

本章内容：

- 介绍 Ajax 异步请求的原理
- 介绍 jQuery 如何让使用 Ajax 技术易如反掌

7.1　Ajax 异步请求原理

7.1.1　Ajax 技术应用程序模型

传统的 Web 应用程序(如普通的 JSP 程序)的运行过程是：发送请求给服务器；服务器对请求进行处理(此时客户端须等待)；处理完成后服务器发送回全新的页面，如图 7-1 所示。

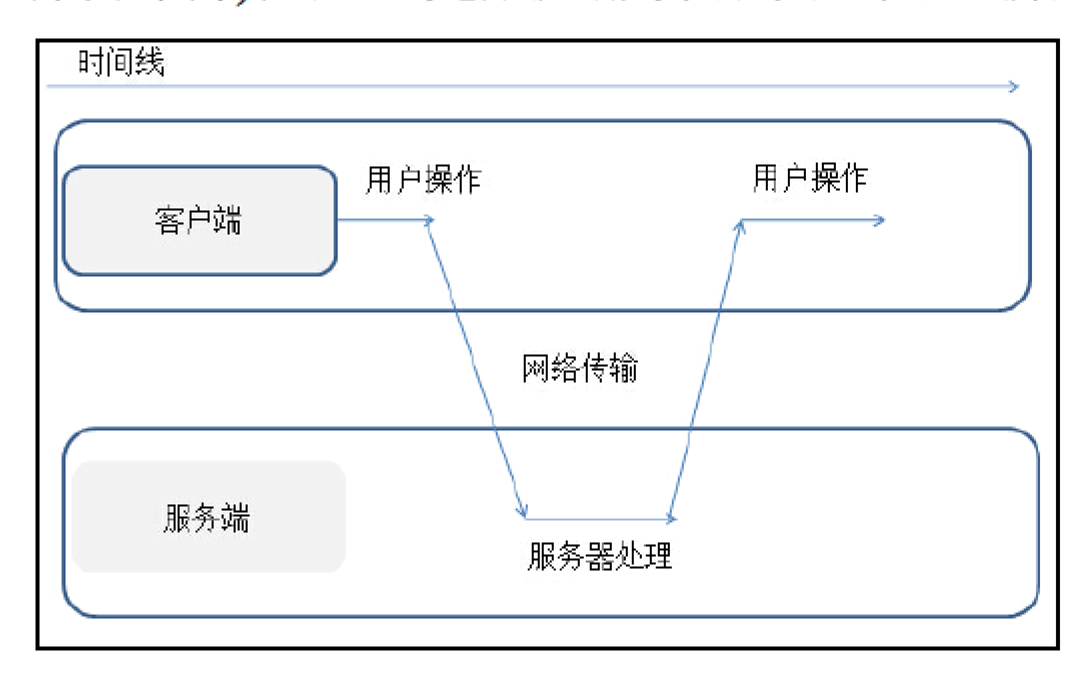


图 7-1　传统的 Web 应用程序

为了加快页面打开的速度并降低服务器的压力，人们想出了一种方案，就是在与 Web 服务器交互的过程中只传输页面上须做更改的区域，而不传输整个页面，这样使传输的数据大大减少，从而缩短了传输时间；同时，在与 Web 服务器交互的过程中，客户端仍然可以在当前页面继续操作，正常使用应用程序，而不必等待服务器的响应。这就是 Ajax 技术的思想，它极大地优化了用户的体验。Ajax 的工作原理是：客户端浏览器在运行时首先加载一个 Ajax

引擎(该引擎由 JavaScript 编写)；Ajax 引擎创建一个异步调用的对象，向 Web 服务器发出一个 HTTP 请求；服务器端处理请求，并将处理结果以 XML 的形式返回；Ajax 引擎接收返回的结果，并通过 JavaScript 语句显示在浏览器上。Ajax 的核心是 JavaScript 对象 XmlHttpRequest，XmlHttpRequest 使得开发人员可以使用 JavaScript 向服务器提出请求并处理响应。Ajax 的 Web 应用程序模型如图 7-2 所示。

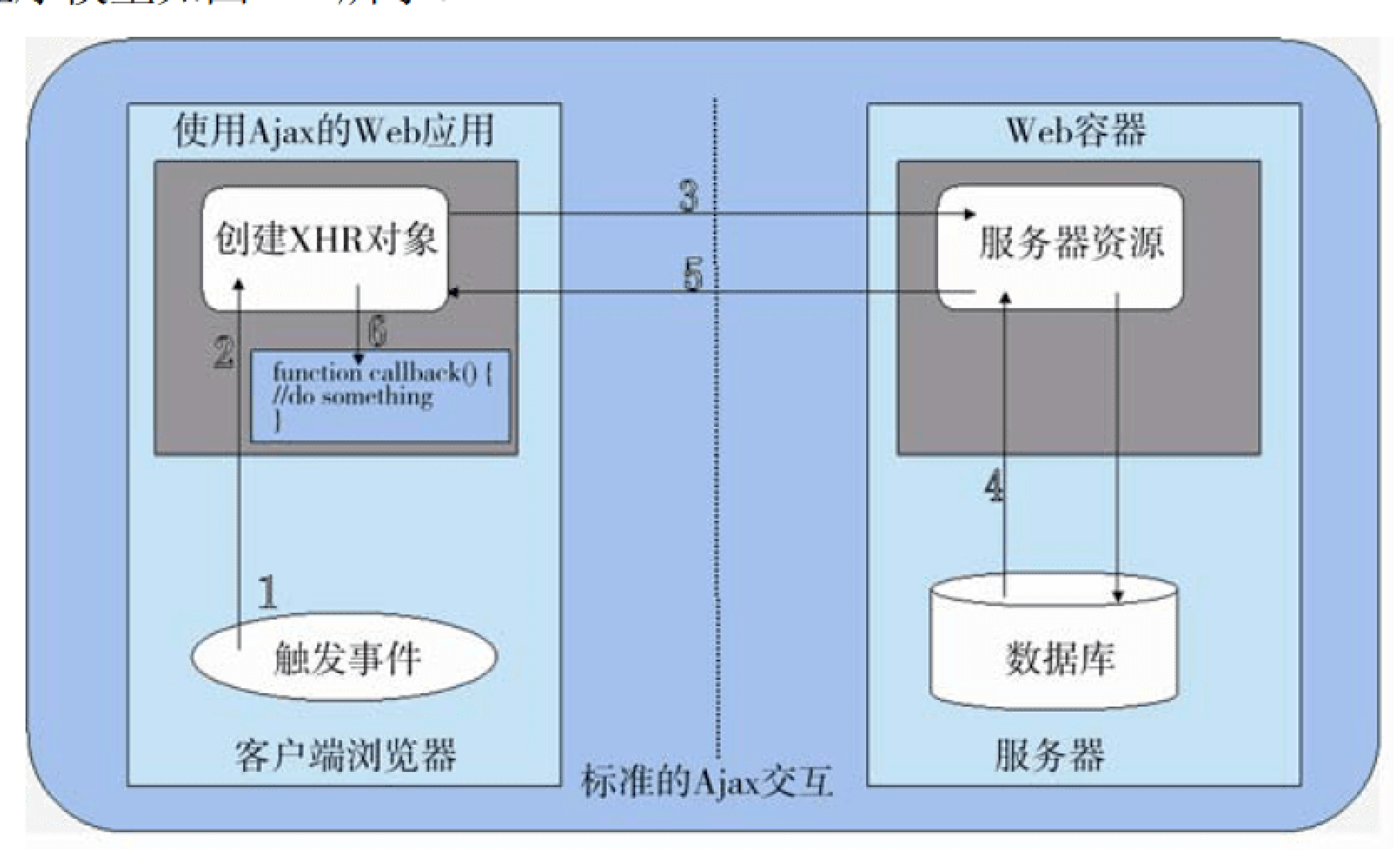


图 7-2　Ajax 的 Web 应用程序模型

7.1.2　使用原始的 Ajax 与服务器通信

一个 Ajax 请求的生命周期包括：创建并设置 XMLHttpRequest 对象、发起请求、保持跟踪状态、获取响应。

1. 创建并设置 XMLHttpRequest 对象

由于 IE6 浏览器使用了 ActiveX 的专有方式，而其他浏览器使用的是标准的 XMLHttpRequest 对象，所以创建 XMLHttpRequest 对象的代码如下：

```
var xmlhttp;
if (window.XMLHttpRequest)
{
  // code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
}
else
{
  // code for IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
```

2. 发起请求

在向服务器发出请求之前，还需要对 XMLHttpRequest 对象做一些设置：设置 HTTP 请

求的类型(GET 或者 POST)、设置请求资源的 URL 地址、设置状态跟踪处理器、设置 POST 请求的发送数据。

XMLHttpRequest 对象的 open()方法指定了请求资源的 http 请求类型、文件名、是否异步方式。代码如下：

```
xmlhttp.open("GET","a.html",true);
```

第三个参数为布尔型，表示是否异步(默认为 true)，毕竟发起同步请求的需求很少。但这个方法不会将请求发送到服务器。

XMLHttpRequest 对象还要准备接收服务器端返回的文档或内容，还要告诉用户请求的状态。XMLHttpRequest 对象的 onreadystatechange 事件可以确切地知道请求当前的状态是什么。设置 onreadystatechange 事件代码如下：

```
xmlhttp.onreadystatechange=function(){}
```

其中绑定的事件处理函数的细节会在下一小节详述。

发送请求的最后一步是将请求对象发送出去，这个步骤是通过 send()方法实现的。如果是 GET 请求方式，代码如下：

```
xmlhttp.send(null);
```

如果是 POST 请求方式，需要在 send()方法中设置我们希望发送的数据。例如 ：

```
xmlhttp.send('a=1&b=2&c=3');
```

3. 保持跟踪状态

onreadystatechange 状态处理函数可以通知页面请求的进度。一旦 send()方法发出请求后，随着请求在不同状态之间转变，此函数会被调用多次。readyState 属性表示服务器对当前请求的处理状态，在状态处理函数中可以根据这个值来进行不同的处理。readyState 有 5 种取值：0 代表尚未初始化；1 代表正在加载；2 代表加载完毕；3 代表正在处理；4 代表处理完毕。例如：

```
xmlhttp.onreadystatechange=function()
{
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
   {
   //请求成功，处理响应数据
   var  message = xmlHttp.responseText;
   }else{
   //错误处理
   }
}
```

一旦 readyState 属性的值变成 4，就表明服务器已经处理完毕。还需要判断 status 属性的值以确定请求是否成功。status 的全称是 HTTP 状态码(HTTP Status Code)，用以表示网页服

务器 HTTP 响应状态的 3 位数字代码。以 2 开头的状态码，代表请求已成功地被服务器接收、理解并接受；以 3 开头的状态码代表需要客户端采取进一步的操作才能完成请求，通常以 3 开头的状态码用来重定向；以 4 开头的状态码代表客户端看起来可能发生了错误，妨碍了服务器的处理；以 5 开头的状态码代表服务器在处理请求的过程中有错误或者异常状态发生。例如：200(请求已成功)、302(请求的资源临时转移到不同的 URI)、404(服务器找不到请求的网页)、410(已删除)、500(服务器端的源代码出现错误)、502(错误网关)。

4. 获取响应

当 readyState 属性的值变为 4 并且 status 属性的值为 200 时，表明服务器已经处理完毕并且没有发生错误，这时就可以处理服务器返回的数据了。Ajax 请求的响应不一定是 XML 格式，它可以是普通的文本、HTML 片段、JSON 格式。不论是哪种格式，都可以使用 responseText 属性获取数据。还可以使用 responseXML 获取数据，但只有响应内容类型头为“text/xml”时，responseXML 才会被解析成为一个 XML 文档。例如：

```
xmlhttp.onreadystatechange=function()
{
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
        //请求成功，处理响应数据
        var xmlDoc = xmlhttp.responseXML.documentElement;
        var xSel = xmlDoc.getElementsByTagName("select");
    }else{
        //错误处理
    }
  }
```

下面的例子将上面的步骤合并为一个例子：

```
<html>
<head>
<meta charset="utf-8">
<script>
function ajaxLoad(){
    var xmlhttp;
    if (window.XMLHttpRequest)
    {
          xmlhttp=new XMLHttpRequest();
    }
    else
    {
          xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.onreadystatechange=function()    {
        if (xmlhttp.readyState==4 && xmlhttp.status==200)
        {
            document.getElementById("myDiv").innerHTML=xmlhttp. responseText;
```

```
            }else{
              console.log("ajax onreadystatechange error");
            }
        }
        xmlhttp.open("GET","loadResourse.html",true);
        xmlhttp.send();
    }
</script>
</head>
<body>
<h2>AJAX</h2>
<button type="button" onclick="ajaxLoad()">请求数据</button>
<div id="myDiv"></div>
 </body>
</html>
```

可见 Ajax 技术的主要内容包括：基于 Web 标准 HTML+CSS 的表示；使用 DOM(Document Object Model)进行动态显示及交互；使用 XML 和 JSON 进行数据交换及相关操作；使用 XMLHttpRequest 进行异步数据加载；使用 JavaScript 将所有的东西绑定在一起。

7.2　jQuery 中载入文档

jQuery 封装了 7.1.2 节中异步提交的代码，简化了我们的操作，同时能够兼容各个浏览器的差异。jQuery 提供了多个与 Ajax 有关的方法，包括 load()方法、$.getJSON()方法、$.getScript()方法、$.get()方法、$.post()方法、$.ajax()方法等。

7.2.1　load()方法载入文档

使用 load()方法可以轻松地从服务器端加载内容，它的语法为：

```
load(url,[data],[callback])
```

其中 url 表示服务器资源的 URL，可以在后面添加选择器。data 表示作为请求参数传递的任何数据，可以是字符串也可以是对象。callback 表示加载成功后执行的回调函数。7.1.2 节的例子我们可以用 load()方法实现，代码如下：

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>load demo</title>
  <script src="https://code.jQuery.com/jQuery-2.1.1.js"></script>
</head>
<body>
<h2>AJAX</h2>
<button type="button" id="projects">请求数据</button>
<div id="myDiv"></div>
```

```
<script>
$( "projects" ).load( "./loadResourse.html" );
</script>
</body>
</html>
```

其中 loadResourse.html 的代码为：

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Sample Page</title>
</head>
<body>
<h1>jQuery 技术</h1>
<ul id="projects">
    <li>jQuery</li>
    <li>jQuery UI</li>
    <li>jQuery Mobile</li>
</ul>
</body>
</html>
```

如果要筛选 load()方法返回的元素，jQuery 允许在 URL 上指定选择器，只要在 URL 后添加紧随空格后的选择器就可以，例如：

```
$( "projects" ).load( "./loadResourse.html #projects" );
```

7.2.2　JSON 格式

JSON(JavaScript Object Notation)是一种轻量级的数据交换格式，它基于 JavaScript 标准，采用完全独立于语言的文本格式。这些特性使 JSON 成为理想的数据交换语言，易于人阅读和编写，同时也易于机器解析和生成。

JSON 语法是 JavaScript 对象表示语法的子集。语法如下：

- 数据在名称/值对中
- 数据由逗号分隔
- 花括号保存对象
- 方括号保存数组

例如：

```
{"programmers":[
{"firstName":"san","lastName":"zhang","email":"aaaa"},
{"firstName":"si","lastName":"li","email":"bbbb"},
{"firstName":"wu","lastName":"wang","email":"cccc"}
],
```

```
"authors":[
{"firstName":"yao","lastName":"lu","genre":" fantasy"},
{"firstName":"Tad","lastName":"Williams","genre":" sciencefiction"},
{"firstName":"Frank","lastName":"Peretti","genre":"christianfiction"}
]
}
```

JSON 值可以是：

- 数字(整数或浮点数)
- 字符串(在双引号中)
- 逻辑值(true 或 false)
- 数组(在方括号中)
- 对象(在花括号中)
- null

可以使用内置的 JSON 对象把一个字符串转换为 JSON 对象，例如：

```
JSON.parse('{"1": 1, "2": 2, "3": {"4": 4, "5": {"6": 6}}}'
```

输出的对象结构如图 7-3 所示。

```
▼Object {1: 1, 2: 2, 3: Object}
   1: 1
   2: 2
 ▼3: Object
     4: 4
   ▼5: Object
       6: 6
```

图 7-3　输出的对象结构

注意：使用 JSON.parse()方法时单引号写在{}外，每个属性名都必须用双引号，否则会抛出异常。下面把一个对象转换为字符串，例如：

```
var a = {a:1,b:2};
JSON.stringify(a);
//输出"{"a":1,"b":2}"
```

7.2.3　比较 JSON 和 XML 数据格式

XML 是可扩展标记语言(extensible markup language)，一种类似于 HTML 的语言，它没有预先定义的标签，使用 DTD(document type definition，文档类型定义)来组织数据，并且格式统一，跨平台和语言，早已成为业界公认的标准。XML 的优点是格式统一、符合标准，容易与其他系统进行远程交互，数据共享比较方便，还是企业服务总线(ESB)方案中的理想选择。但其也有缺点，包括：XML 文件庞大，格式复杂，传输占用带宽；服务器端和客户端都需要花费大量代码来解析 XML，使得服务器端和客户端代码变得异常复杂和不容易维护，需要重复编写很多代码；服务器端和客户端解析 XML 花费较多资源和时间。

JSON 也用来表示文本数据，也可以跨平台。它的优点包括：数据格式比较简单、易于读写、占用带宽小、浏览器解析速度快；构造友好、支持多种语言；JSON 格式能够直接为服务器端代码使用，大大简化了服务器端和客户端的代码开发量。其缺点包括：没有 XML 那么通用，在 Web Service 中推广还属于初级阶段。

本书接下来的程序大多采用 JSON 数据格式与服务器进行通信。

下面介绍人人微博网站中后台提供的 API，返回的数据也都是 JSON 格式。

1. 新增用户

接口说明：新增用户。

URL：http://localhost:8080/rrwb/crud/user/add。

HTTP 请求方式：POST/GET。

请求参数如表 7-1 所示。

表 7-1　请求参数

参 数 名 称	类　型	说　明
login_name	String	登录名，必输
nick_name	String	昵称，必输
img	String	缩略图地址
real_name	String	真实姓名
provinceValue	String	所在省份编号
provinceName	String	所在省份名称
cityValue	String	所在城市编号
cityName	String	所在城市名称
gender	String	性别
age	String	年龄
other_email	String	备用邮箱
blog	String	博客地址
qq	String	QQ
pwd	String	密码，必输
msn	String	MSN
birthday	String	生日

响应参数如表 7-2 所示。

表 7-2　响应参数

参 数 名 称	类　型	描　述
code	Number	0：插入失败 1：插入成功

2. 根据 id 查询用户

接口说明：根据用户 id 查找用户。

URL：http://localhost:8080/rrwb/crud/user/edit。

HTTP 请求方式：POST/GET。

请求参数如表 7-3 所示。

表 7-3　请求参数

参 数 名 称	类　型	说　明
_id	Number	用户 id，必输

响应参数如表 7-4 所示。

表 7-4　响应参数

参 数 名 称	类　型	描　述
code	Number	0：插入失败 1：插入成功

3. 根据用户登录名查找用户

接口说明：根据用户登录名查找用户。

URL：http://localhost:8080/rrwb/crud/user/findUserByName。

HTTP 请求方式：POST/GET。

请求参数如表 7-5 所示。

表 7-5　请求参数

参 数 名 称	类　型	说　明
login_name	String	登录名，必输

响应参数如表 7-6 所示。

表 7-6　响应参数

参 数 名 称	类　型	描　述
code	Number	0：插入失败 1：插入成功

4. 根据用户昵称查找用户

接口说明：根据用户昵称查找用户。

URL：http://localhost:8080/rrwb/crud/user/ findUserByNickName。

HTTP 请求方式：POST/GET。

请求参数如表 7-7 所示。

表 7-7　请求参数

参 数 名 称	类　型	说　明
nick_name	String	昵称，必输

响应参数如表 7-8 所示。

表 7-8　响应参数

参 数 名 称	类　型	描　述
code	Number	0：插入失败 1：插入成功

5. 根据用户昵称模糊查找用户

接口说明：根据用户昵称模糊查找用户，并可以分页查询。

URL：http://localhost:8080/rrwb/crud/user/findUserLikeNickName。

HTTP 请求方式：POST/GET。

请求参数如表 7-9 所示。

表 7-9　请求参数

参 数 名 称	类　型	说　明
nick_name	String	昵称，必输
_id	Number	用户 id，必输
page	String	当前页数，从 1 开始
size	String	每页显示的数量

响应参数如表 7-10 所示。

表 7-10　响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	String	查询到的数据的数组
rows	String	查询到的记录总数量

6. 用户登录

接口说明：根据用户名和密码验证用户登录。

URL：http://localhost:8080/rrwb/crud/user/ login。

HTTP 请求方式：POST/GET。

请求参数如表 7-11 所示。

表 7-11　请求参数

参 数 名 称	类　型	说　明
login_name	String	登录名，必输
pwd	String	密码，必输

响应参数如表 7-12 所示。

表 7-12　响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
data	String	查询到的数据的数组
rows	String	查询到的记录总数量

例如：

```
{"code":1,"pages":1,"data":[{"_id":103,"login_name":"1@1.com","gender":
 "female","nick_name":"hello","pwd":"111111","status":true,"timestamp":
 1427088263009},{"_id":92,"login_name":"wyy@163.com","gender":"female",
"nick_name":"wuyuan","pwd":"111111","status":true,"timestamp":1426058627885},
{"_id":91,"login_name":"zp@163.com","gender":"female","nick_name":"zhangsan",
 "pwd":"111111","status":true,"timestamp":1426058555856}],"rows":3}
```

7.2.4　$.getJSON()方法载入文档

$.getJSON()方法的语法为：

```
$.getJSON(url,data,success(responseData,status,xhr))
```

其中 url 表示服务器资源的 URL，可以在后面添加选择器。data 表示作为请求参数传递的任何数据，可以是字符串也可以是对象。success 表示加载成功后执行的回调函数，回调函数的第一个参数是返回的数据，第二个参数是字符串表示的请求状态("success"、"notmodified"、"error"、"timeout"、"parsererror")，第三个参数表示 XMLHttpRequest 对象。比如有一个名为 test.json 的文件，内容如下：

```
{
  "one": "一步"
  "two": "两步"
  "three": "三步"
}
```

那么使用$.getJSON()方法就可以在网页中加载这个 JSON 文档。由于加载的 JSON 数据不能在页面中直接显示，success 回调函数的作用就是处理 JSON 数据使其在页面上显示。success 回调函数的第一个参数 data 代表返回的数据。代码如下：

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Sample Page</title>
      <script src="https://code.jQuery.com/jQuery-2.1.1.js"></script>
</head>
<body>
</body>
<script>
```

```
$.getJSON( "test.json", function( data ) {
  var items = [];
  $.each( data, function( key, val ) {
    items.push( "<li id='" + key + "'>" + val + "</li>" );
  });
  $( "<ul/>", {
    html: items.join( "" )
  }).appendTo( "body" );
});
</script>
</html>
```

上面的代码将加载的数据以列表的形式追加到 body 中。在 jQuery 1.5 之后，允许在$.getJSON()之后链式调用.done()、.always()和 .fail()函数，例如：

```
$.getJSON( "test.json").done(function( data ) {
    alert( "Data Loaded: " + data );
}).fail(function() {
    console.log("error");
 }).always(function() {
    console.log("complete");
 });
```

下面我们来实现注册时验证登录名和昵称是否重复的功能，代码如下：

```
$("#login_name").blur(function(event) {
     var login_name = $("#login_name").val();
     var login_name_checkOk = false;
     if(check_mobile(login_name) || check_email(login_name)){
       login_name_checkOk = true;
     }
     if(!login_name_checkOk){
       $("#login_name").val('');
       alert("格式不正确");
       return;
     }
       //检查用户登录名
       $.getJSON('http://localhost:8080/rrwb/crud/user/findUserByName?
            login_name='+login_name).done(function(response) {
         if(response.code == "1"){
             if(response.data.length>0){
                 $("#login_name").val('');
                 alert("已被占用！");
             }
         }
       })
       .fail(function() {
          console.log("error");
       })

       });
```

```
$("#password").blur(function(event) {
    var num=$("#password").val().length;
      if(num<6){
        $("#password").val('');
        alert("密码小于 6 位");

      }else if(num>18){
        $("#password").val('');
        alert("密码大于 18 位");
      }else {

      }
});

$("#re_password").blur(function(event) {
    var num=$("#re_password").val().length;
      if(num<6){
        $("#re_password").val('');
        alert("密码小于 6 位");

      }else if(num>18){
        $("#re_password").val('');
        alert("密码大于 18 位");
      }
        var pass=$("#password").val();
        var re_pass=$("#re_password").val();
        if(pass!=re_pass){
          $("#re_password").val('');
          alert("密码不一致");
        }
});
 $("#nickname").blur(function(event) {
    var nickname=$("#nickname").val();
    if(nickname.length==0){
        alert("请输入昵称");
    }else{
 //检查用户昵称
 $.getJSON('http://localhost:8080/crud/user/findUserByNickName?nick
 _name='+nickname).done(function(response) {
  if(response.code == "1"){
      if(response.data.length>0){
          $("#nickname").val('');
          alert("已被占用！");
      }
   }
})
.fail(function() {
   console.log("error");
```

```
        })
      }
  });
});
```

7.2.5　$.getScript()方法载入文档

当网站需要加载大量 js 时，动态地加载 js 就是一个比较好的方法，当需要某个功能时再将相应的 js 加载进来。jQuery 提供了$.getScript()方法获取 js 文件，可以像加载页面一样注入脚本，并且所注入的脚本自动执行。它的语法为：

```
$.getScript (url,success(data,status,xhr))
```

例如：

```
$.getScript( "test.js", function( data, textStatus, jqxhr ) {
  console.log( data ); // Data returned
  console.log( textStatus ); // Success
  console.log( jqxhr.status ); // 200
  console.log( " getScript was performed." );
});
```

test.js 的内容中只有一行代码：

```
console.log("in test.js");
```

输出结果为：

```
in test.js
console.log("in test.js");
success
200
getScript was performed.
```

7.3　jQuery 中请求服务器数据

7.3.1　$.post()方法

$.post()方法不仅可以载入远程页面和加载数据，还可以给远程页面发送数据，在发送数据的时候是以 POST 方式发送的。它的语法为：

```
$.post( url [, data ] [, success ] [, dataType ] )
```

其中有些参数可以省略，与之相应的逗号也可以省略。各个参数的说明如表 7-13 所示。

表 7-13　$.post()方法的参数

参数名称	类　型	说　明
url	String	请求资源的 URL 地址
data	Object	发送给服务器的 key:value 数据
success	Function	回调函数，载入成功后会执行的函数
type	String	服务器端返回的内容的格式，可以是 html、json、xml、script、jsonp、text 等

下面我们来实现注册时提交注册信息到服务器的功能，代码如下：

```
function regist(){
          var login_name = $("#login_name").val();
          var password = $("#password").val();
          var re_password = $("#re_password").val();
          var nickname = $("#nickname").val();
          var gender = $("input[type='radio']:checked").val();
          if(login_name.length>0&&password.length>0&&re_password.length>
              0&&nickname.length>0&&gender.length>0){
            $.post('http://localhost:8080/rrwb/crud/user/add',
              {"login_name": login_name,"pwd":re_password,"gender":gender,
              "nick_name":nickname},
                function(response) {
                  if(response.code == "1"){
                    //设置注册按钮不可用
                    $('a').attr("disabled","disabled");
                    alert("注册成功！");
                    //转到登录页面
                    location.href="index.html"
                    console.log("success");
                  }
                        },
              'json'
            )
          }
      }
```

7.3.2　$.get()方法

$.get()方法与$.post()方法的用法相似，区别在于发送数据时以 POST 方式发送。语法为：

```
$.get( url [, data ] [, success ] [, dataType ] )
```

其中参数的意义与$.post()方法的参数意义相同。这里使用$.get()方法实现登录功能，代码如下：

```
$(document).ready(function() {
    $( "#login_btn" ).click(function(event) {
    var login_name=document.getElementById("login_name");
    var password=document.getElementById("password");
```

```
        login_name.value=login_name.value.replace(/\s{1,}/g,"");

        if (login_name.value==""){
            alert("请输入账户！");
            login_name.focus();
            return;
        }
        if (password.value==""){
            alert("请输入密码！");
            password.focus();
            return;
        }
        $.get(
            'http://localhost:8080/rrwb/crud/user/login',
            {login_name: login_name.value,pwd:password.value},
            function(response) {
                if(response.rows == 1 ){
                        //使用 sessionStorage 存储 userId
                    sessionStorage.userId=response.data[0]._id;
                        //转向个人主页
                    location.href = "home.html";
                }else{
                    event.preventDefault();
                    alert("用户名或密码错误");
                }
            },
            'json'
        );
    });
});
```

7.3.3　表单的序列化方法

在表单中的输入框较少的情况下，可以通过名称属性逐个搜索输入字段来传输数据，如果表单的输入字段过多，这种方式就比较麻烦，而且缺乏灵活性。为了解决这个问题，jQuery 引入了 serialize()方法，该方法可以简化参数传值的方式，其调用的语法格式如下:

```
$(selector).serialize()
```

通过该方法将创建一个以标准 URL 编码方式表示的文本字符串并返回该字符串，在 Ajax 请求中可将该字符串发送给服务器。例如：

```
<form>
    <!-- 能够被序列化的元素类型 -->
     <input type="text" name="a" value="1" id="a" />
     <input type="text" name="b" value="2" id="b" />
     <input type="hidden" name="c" value="3" id="c" />
     <textarea name="d" rows="8" cols="40">4</textarea>
     <select name="e">
```

```
            <option value="5" selected="selected">5</option>
            <option value="6">6</option>
            <option value="7">7</option>
      </select>
      <input type="checkbox" name="f" value="8" id="f1" checked="checked" />
        <input type="checkbox" name="f" value="9" id="f2" />
        <input type="radio" name="g" value="10" id="g1"/>
      <input type="radio" name="g" value="11" checked="checked" id="g2"/>
        <input type="password" name="h" maxlength="8" value="h" />
     <!-- 不能够被序列化的元素类型 -->
        <input type="submit" name="i" value="Submit" id="i" />
        <input type="button" name="j" value="Click me" onclick="msg()" />
        <input type="file" name="k" /><br />
        <input type="reset" value="Reset"  name="l" />
        <input type="image" src="" alt="Submit" />
</form>
<p><tt id="results1"></tt></p>
<p><tt id="results2"></tt></p>
<script>
    function showValues() {
      var str = $("form").serialize();
      $("#results2").text(str);
    }
    $(":checkbox, :radio").click(showValues);
    $("select").change(showValues);
    showValues();
</script>
```

上面的代码的显示结果为“a=1&b=2&c=3&d=4&e=5&f=8&g=11&h=h”。

此外还有一个方法可以序列化表单，唯一的区别在于该方法不是返回字符串，而是将一组表单元素编码为一个名称和值的 JSON 数组对象。这个方法就是 serializeArray()，语法为：

```
$(selector).serializeArray ()
```

序列化后得到的对象格式如下所示：

```
[
  {name: 'firstname', value: 'Hello'},
  {name: 'lastname', value: jQuery},
  {name: 'alias'},
]
```

可以选择一个或多个表单元素，比如 input、textarea 或者 form 元素本身。

同样是 serialize()方法中的示例，如果使用的是 serializeArray()方法序列化表单，jQuery 代码如下：

```
<script>
    function showValues() {
```

```
        var fields = $(":input").serializeArray();
        $("#results").empty();
        jQuery.each(fields, function(i, field){
          $("#results1").append(field.value + " ");
        });
      }
      $(":checkbox, :radio").click(showValues);
      $("select").change(showValues);
      showValues();
</script>
```

上面的代码的显示结果为“a:1　b:2　c:3　d:4　e:5　f:8　g:11　h:h”。

serialize()方法和 serializeArray()方法使用了 W3C 关于 successful controls(有效控件)的标准来检测哪些元素应当包括在内。特别说明，元素不能被禁用(禁用的元素不会被包括在内)，并且元素应当含有 name 属性。提交按钮的值也不会被序列化，文件选择元素的数据也不会被序列化。

Ajax 的出现带来了单页 Web 应用(single-page application，SPA)，比如 Gmail。单页 Web 应用为了充分利用 Ajax，必须在单一页面中包含其中的全部功能，或者是至少包含大部分功能，浏览器与 Web 应用程序的所有交互只能在一个页面的范围内进行。此方法对 Web 来说是一种创新，它可以像 Windows 桌面程序一样，独立加载、更新和替换一些可视元素的组合。

有时单页 Web 应用是一种从 Web 服务器加载的胖客户端应用。

胖客户端是相对于瘦客户端而言的，瘦客户端应用程序中浏览器只解析标准的 HTML 来显示用户交互界面。这样，服务器端负责处理业务逻辑和数据存取，然后将处理完的结果以 HTML 的形式发送给客户端，客户端负责将结果显示给用户。客户端除了负责一些数据的验证和组织之外，基本上不处理任何业务逻辑，只专注于用户交互界面的显示。

胖客户端应用程序的客户端部分除了负责将程序的 UI 界面显示给用户进行交互外，还负责大部分的业务逻辑处理。这种类型的应用程序需要客户端部分具有执行任务的能力，对客户端机器的要求比较高，但是可以减轻服务器很大一部分压力，降低对服务器性能的要求。典型的胖客户端应用程序拥有丰富的交互式用户界面，用户通过这个交互界面可以查看和操作数据、处理业务事务等，分担服务器的一部分或者全部业务逻辑的处理。

使用 Ajax 构建胖客户端是一个最严峻的挑战。胖客户端可以是分布式企业系统的前端，也可以是业务线应用程序的表现逻辑层。这些应用程序无论是发布到 Internet 上还是限制在 intranet(内部网络)内，都需要常规桌面 UI 所具有的丰富性和速度。单页 Web 应用会包含大量的 JavaScript 代码，复杂度可想而知，模块化开发和设计的重要性不言而喻。单页 Web 应用也有一些缺点，所有的内容都在一个页面中动态替换显示，对搜索引擎优化不友好。由于单页 Web 应用在一个页面中显示所有的内容，所以不能使用浏览器的前进后退功能，所有的页面切换需要自己建立堆栈管理。

7.4　$.ajax()方法

7.4.1　详解$.ajax()的细节

jQuery 提供了更底层的 Ajax 通信方法，我们可以利用 jQuery 提供的$.ajax()方法控制 Ajax 请求的各种细节。$.get 和$.post 方法最终都是使用这个函数发起。它的语法为：

```
$.ajax([options])
```

它的参数 options 是一个对象，其格式为 key/value。它可以定义的值非常多，具体见表 7-14。

表 7-14　$.ajax()方法的参数

参　　数	类　　型	说　　明
url	String	发送请求的地址
type	String	要使用的 HTTP 方法。通常是 GET 或 POST。默认为 GET
data	String	发送到服务器的数据。将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。无论哪种情况，$.ajax()函数都会负责对值进行编码
dataType	String	预期服务器返回的数据类型。如果不指定，jQuery 将自动根据 HTTP 包 MIME 信息返回 responseXML 或 responseText，并作为回调函数参数传递，可用值包括： "xml"：返回 XML 文档，可用 jQuery 处理。 "html"：返回纯文本 HTML 信息；包含 script 元素。 "script"：返回纯文本 JavaScript 代码。不会自动缓存结果。 "json"：返回 JSON 数据。 "jsonp"：JSONP 格式。使用 JSONP 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换?为正确的函数名，以执行回调函数
cache	String	设置为 false 将不会从浏览器缓存中加载请求信息。默认为 true，除非指定 dataType 为 script 或 jsonp
context	Object	指定某个元素为这个请求的所有回调函数的上下文
timeout	Number	设置请求超时时间(毫秒)。此设置将覆盖全局设置
global	String	是否触发全局 Ajax 事件，默认为 true。设置为 false 将不会触发全局 Ajax 事件，如 ajaxStart 或 ajaxStop。可用于控制不同的 Ajax 事件
contentType	String	发送信息至服务器时的内容编码类型。默认值为"application/x-www-form-urlencoded"，适合大多数应用场合
error	String	请求失败时将调用此方法。这个方法有三个参数：XMLHttpRequest 对象、错误信息、(可能)捕获的错误对象

(续表)

参　　数	类　　型	说　　明
complete	String	请求完成后回调函数(请求成功或失败时均调用)。参数：XMLHttpRequest 对象、成功信息字符串(success 或 error)
beforeSend	String	发送请求前可修改 XMLHttpRequest 对象的函数，如添加自定义 HTTP 头
async	Boolean	默认设置下(默认为 true)，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。注意，同步请求将锁住浏览器，用户的其他操作必须等待请求完成才可以执行
processData	String	默认情况下(默认为 true)，发送的数据将被处理为 URL 编码格式，适合与 "application/x-www-form-urlencoded"一起使用。如果要发送 DOM 树信息或其他不希望转换的信息，请设置为 false
ifModified	Boolean	如果设置为 true，则只有当响应内容相对于上次请求改变(根据 Last-Modified 头设置)时，请求才被认为是成功的。默认是 false
username	String	在 HTTP 认证请求中使用的用户名
password	String	在 HTTP 认证请求中使用的密码
Scriptcharset	String	当远程和本地内容使用不同的字符集时，用来设置 script 和 jsonp 请求所使用的字符集
jsonp	String	指定一个查询参数名称覆盖默认的参数名 callback
jsonpCallback	String	指定 jsonp 回调函数名。默认是随机生成的函数名

在使用 load 方法加载 html 的例子中，同样的操作如果用$.ajax()来实现，代码如下所示：

```
$.ajax({
    url: './loadResourse.html ',
    method: 'GET',
    dataType:'html'
    success: function (data) {
        $("#new-projects" ).append(data);
    }
});
```

$.ajax()函数依赖服务器提供的信息来处理返回的数据。如果服务器报告说返回的数据是 XML，那么返回的结果就可以用普通的处理 XML 的方法或者 jQuery 的选择器来遍历。如果得到其他类型的数据，比如 HTML，则数据就以文本形式来对待。

通过 dataType 选项还可以指定其他不同数据处理方式。除了单纯的 XML，还可以指定 html、json、jsonp、script 或 text。其中，text 和 xml 类型返回的数据不会经过处理，数据仅仅将 XMLHttpRequest 的 responseText 或 responseHTML 属性传递给 success 回调函数。如果指定为 html 类型，任何内嵌的 JavaScript 都会在 HTML 作为一个字符串返回之前执行。类似地，指定 script 类型的话，也会先执行服务器端生成 JavaScript，然后再把脚本作为一个文本数据返回。如果指定为 json 类型，则会把获取到的数据作为一个 JavaScript 对象来解析，并且把构建好的对象作为结果返回。为了实现这个目的，它首先尝试使用 JSON.parse()。

如果指定了 script 或者 jsonp 类型，那么当从服务器接收到数据时，实际上是用了<script>标签而不是 XMLHttpRequest 对象。这种情况下，$.ajax()不再返回一个 XMLHttpRequest 对象，并且也不会传递事件处理函数，比如 beforeSend。加载其他网域的数据的时候涉及跨域的概念，域(Domain)是网络中独立运行的单位，域之间相互访问则需要建立信任关系(Trust Relation)。只要协议、域名、端口有任何一个不同，都被当作是不同的域。有一种简明的说法来解释跨域：跨域访问简单来说就是 A 网站的 JavaScript 代码试图访问 B 网站，包括提交内容和获取内容。由于安全原因，跨域访问是被各大浏览器所默认禁止的。

虽然浏览器默认禁止了跨域访问，但并不禁止在页面中引用其他域的 JS 文件，并可以自由执行引入的 JS 文件中的函数，这也是 JSONP 的由来。JSONP 是 JSON with Padding 的简称(Padding 在这里理解为填充，JSONP 也叫填充式 JSON)。它是一个非官方的协议，它通过 JavaScript callback 的形式实现跨域访问，是应用 JSON 的一种新方法，只不过是被包含在函数调用中的 JSON 应用。例如：

```
<script type="text/javascript">
   function dosomething(jsondata){
      //处理获得的 json 数据
   }
</script>
<script src="http://example.com/data.php?callback=dosomething"></script>
```

js 文件载入成功后会执行我们在 url 参数中指定的函数，并且会把我们需要的 json 数据作为参数传入。所以 jsonp 是需要服务器端的页面进行相应配合的。下面是服务器端代码：

```
<?php
$callback = $_GET['callback'];//得到回调函数名
$data = array('a','b','c');//要返回的数据
echo $callback.'('.json_encode($data).')';//输出
?>
```

最终，输出结果为“dosomething(['a','b','c']);”。

jQuery 会自动生成一个全局函数来替换 callback=?中的问号，在获取到数据后又会自动销毁。比如我们使用 jQuery 跨域访问 github.com 的 API，代码如下：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="https://code.jQuery.com/jQuery-2.1.1.js"></script>
<script>
    //如果去掉 callback=? ， 则不能访问数据
    $.getJSON('https://api.github.com/users/musicpop?callback=?',
    function(result) {
            for(var i in result.data) {
                console.log(i+":"+result.data[i]);
      }
    });
```

```
</script>
</head>
</html>
```

输入结果如图 7-4 所示。

```
login:musicpop
id:1130903
avatar_url:https://avatars.githubusercontent.com/u/1130903?v=3
gravatar_id:
url:https://api.github.com/users/musicpop
html_url:https://github.com/musicpop
followers_url:https://api.github.com/users/musicpop/followers
following_url:https://api.github.com/users/musicpop/following{/other_user}
```

图 7-4　jsonp 示例

在$.ajax()方法的参数中，jsonp 用来指定一个名称，覆盖默认的参数名 callback，jsonpCallback 指定 jsonp 回调函数名。例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="https://code.jQuery.com/jQuery-2.1.1.js"></script>
<script>
    jQuery(document).ready(function(){
        $.ajax({
          type: "get",
          async:true,
          url: "https://api.github.com/users/musicpop?callback=?",
          dataType: "jsonp",
         //传递给请求处理程序或页面，用以获得 jsonp 回调函数名的参数名(默认为:callback)
          jsonp: "callbackparam",
         //自定义的 jsonp 回调函数名称，默认为 jQuery 自动生成的随机函数名
         jsonpCallback:"success_jsonpCallback",
         success: function(result){
           console.log(result);
            for(var i in result.data) {
               console.log(i+":"+result.data[i]);
               }
          },
          error:function(){
              alert('fail');
          }
        });
    });
</script>
</head>
</html>
```

使用$.ajax()方法显然增加了编码的工作量，可是却带来了更大的灵活性。避免浏览器缓存来自服务器的响应。非常适合服务器动态生成数据。可以注册独立的回调函数，制止正常情况下所有 Ajax 交互中的全局事件。在远程主机需要认证的情况下，可以提供用户名和密码。

虽然使用 Ajax 有诸多好处，但是使用 Ajax 时应该遵循一定的原则：

- 尽量减少通信量：Ajax 应用程序向服务器发送的信息量及从服务器接收的信息量应尽可能地少。简单地说，Ajax 应尽量减少客户端和服务器之间的通信流量。
- 确保 Ajax 应用程序不发送和接收不需要的信息。
- 不管选择什么样的用户交互模型，一定要确保用户知道下一步该如何操作。
- 不需要在使用新的交互模型上浪费时间，直接参考传统的 Web 应用程序和桌面应用程序，这样可以使用户学习更快捷。
- 避免下载整个页面：当最初的页面下载之后，所有与服务器的通信都将由 Ajax 引擎管理。

7.4.2　$.ajax()全局事件

在使用$.ajax()方法时可以注册回调函数，包括 beforeSend、success、error、complete，我们称这些事件为局部事件。jQuery 还提供了全局的 Ajax 事件，全局事件可以绑定到 DOM 元素上，这样我们就可以通过改变 DOM 元素来跟踪请求进展的状态。全局事件在被触发后会广播到 document 元素上，从 jQuery1.8 以后，全局事件只能绑定在 document 对象上。当执行函数时上下文被设置为在其上创建处理器的 document 对象。全局事件如表 7-15 所示。

表 7-15　Ajax 全局事件

事件名称	参　数	功能描述
.ajaxSuccess()	Callback	当 Ajax 请求成功时显示一条消息
.ajaxError()	Callback	当 Ajax 请求完成且出现错误时注册要调用的处理程序
.ajaxStart()	Callback	当首个 Ajax 请求开始时执行函数
.ajaxStop()	Callback	当所有 Ajax 请求完成时执行函数
.ajaxSend()	Callback	在一个 Ajax 请求发送之前执行
.ajaxComplete()	Callback	当一个 Ajax 请求完成时执行

例如：

```
<body>
<b>Projects:</b>
<ol id="new-projects"></ol>
<button>改变内容</button>
<script>
$(document).ready(function(){
  $(document).ajaxSend(function(){
    $("body").append("<img src='loading.gif' />");
  });
  $("button").click(function(){
```

```
    $( "#new-projects" ).load( "./loadResourse.html #projects li" );
    $( "#new-projects" ).load( "./loadResourse.html #projects li" );
    $( "#new-projects" ).load( "./loadResourse.html #projects li" );

     });
});
</script>
```

当上面的代码在运行时点击按钮后，会在页面底部增加三个正在加载的图片，如图 7-5 所示。

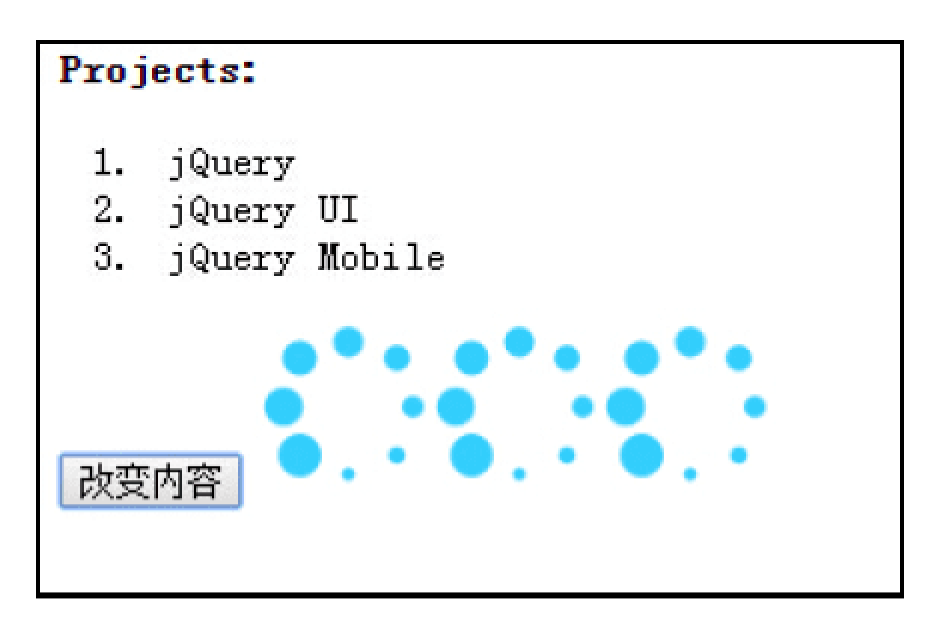


图 7-5　ajaxSend 示例

ajaxStart 与 ajaxSend 的区别是：当一起执行多个 Ajax 请求时 ajaxStart 只会执行一次，也就是当一个 Ajax 请求开始时，如果其他 Ajax 请求处于激活状态，则不会触发 ajaxStart 事件。而 ajaxSend 在每一个 Ajax 请求开始之前都会触发。例如：

```
<body>
<b>Projects:</b>
<ol id="new-projects"></ol>
<button>改变内容</button>
<script>
$(document).ready(function(){
  $(document).ajaxStart(function(){
    $("body").append("<img src='loading.gif' />");
  });
  $("button").click(function(){
    $( "#new-projects" ).load( "./loadResourse.html #projects li" );
    $( "#new-projects" ).load( "./loadResourse.html #projects li" );
    $( "#new-projects" ).load( "./loadResourse.html #projects li" );
  });
});
</script>
```

当上面的代码在运行时点击按钮后，会在页面底部增加一个正在加载的图片，如图 7-6 所示。

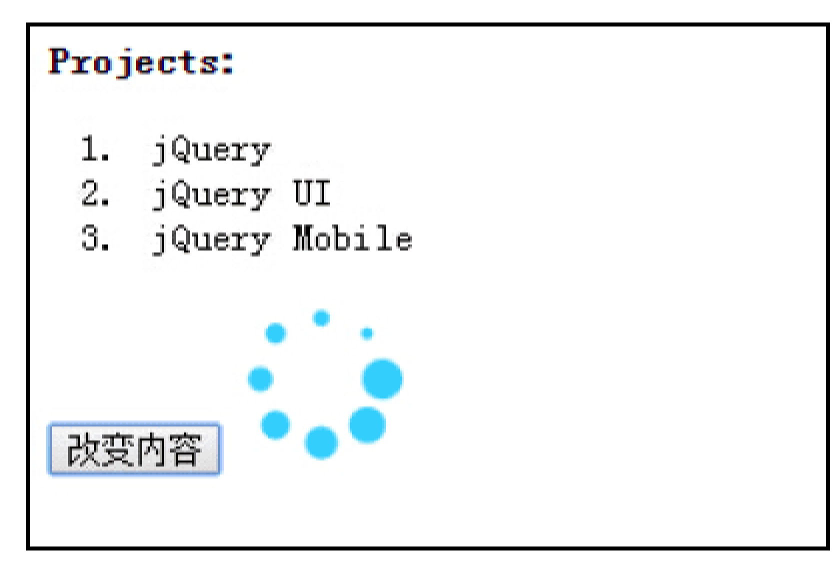


图 7-6　ajaxStart 示例

7.4.3　$.ajax()全局设定

使用 ajaxSetup()方法可以设置 Ajax 请求的一些全局性选项值，设置完成后，后面的 Ajax 请求将不需要再设置这些选项值，它的语法为：

```
$.ajaxSetup([options])
```

可选项 options 参数为一个对象，通过该对象设置 Ajax 请求的全局选项值。这样就不需要对每个$.ajax()方法都设置它的细节，当页面中多个地方都利用$.ajax()方法进行通信时，使用 ajaxSetup()方法简化代码结构。例如：

```
<body>
  <button id="btn1">加载 1</button>
  <button id="btn2">加载 2</button>
</body>
<script>
$(document).ready(function() {
  $.ajaxSetup({
    dataType: 'json',
    success:function(data){
      var items = [];
      $.each( data, function( key, val ) {
        items.push( "<li id='" + key + "'>" + val + "</li>" );
      });
      $( "<ul/>", {
        html: items.join( "" )
      }).appendTo( "body" );
    }
  })
  $("#btn1").click(function(event) {
    $.ajax({url:"https://api.github.com/users/musicpop"});
  });
  $("#btn2").click(function(event) {
    $.ajax({url:"https://api.github.com/users/jrburke"});
  });
});
</script>
```

上面的例子中，两个按钮都可以异步加载数据，并使用了相同的 ajax 设置。

在页面初始化的时候$.ajaxSetup()方法只会运行一次，全局设定不能设置 load()方法的相

关操作，也不能改变$.post()方法采用 POST 方式发送数据。

Ajax 是浏览器技术发展的成果，通过在浏览器端发送异步请求，提高了单用户操作的响应性。但 Web 本质上是一个多用户的系统，对任何用户来说，可以认为服务器是另外一个用户。现有 Ajax 技术的发展并不能解决在一个多用户的 Web 应用中，将更新的信息实时传送给客户端，从而用户可能在“过时”的信息下进行操作。而 Ajax 的应用又使后台数据更新更加频繁成为可能。Ajax 的一个最大优点是可以实现实时页面更新。但是，实时更新如果被误用，可能会成为应用程序的严重威胁。假设某个用户显示了一个活动页面，此页面每隔几秒钟轮询一次服务器以更新一些内容。如果该用户离开几个小时，但没有关闭浏览器，这样做产生的结果是，页面不断发送请求，给服务器带来大量(而且无用)的工作负荷。

于是有了许多对“服务器推”技术的需求，一些技术专家称这种基于 HTTP 长连接、无须在浏览器端安装插件的“服务器推”为 Comet。目前已经出现了一些成熟的 Comet 应用以及开源框架，一些 Web 服务器如 Jetty 也为支持大量并发的长连接进行了很多改进。

所谓长连接，就是要在客户端与服务器之间创建和保持稳定可靠的连接。其实它是一种很早就存在的技术，但是由于浏览器技术的发展比较缓慢，没有为这种机制的实现提供很好的支持。所以要达到这种效果，需要客户端和服务器的程序共同配合来完成。通常的做法是，在服务器的程序中加入一个无限循环，在循环中监测数据的变动。当发现新数据时，立即将其输出给浏览器并断开连接，浏览器在收到数据后，再次发起请求以进入下一个周期，这就是常说的长轮询(long-polling)方式。

如图 7-7 所示，它通常包含以下几个关键过程。

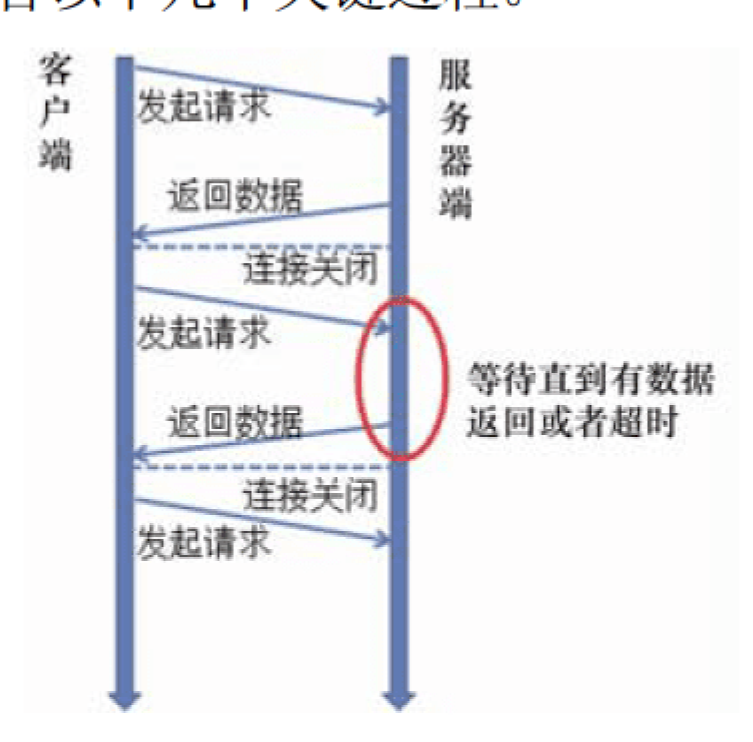


图 7-7　长轮询的几个关键过程

当循环过程中服务器向浏览器推送信息后，应该主动结束程序运行从而让连接断开，这样浏览器才能及时收到数据并发起新的请求。当没有新数据推送时，循环不能一直持续下去，应该设定一个最长时限，避免 Web 服务器超时(Timeout)。若一直没有新信息，服务器应主动向浏览器发送本次轮询无新信息的正常响应，并断开连接。当出现网络故障或异常造成的请求超时或出错时，也可能导致轮询的意外中断，此时浏览器将收到错误信息。

使用 jQuery 可以实现长连接，代码如下：

```
<!DOCTYPE html>
<html>
    <head>
    <meta charset="utf-8" />
```

```
    <script src="rrwb/scripts/jQuery-2.1.1.js"></script>
    <script>
          function fetch() {
             $.ajax({
                url: 'http://localhost:8080/rrwb/crud/longpolling',
                async: true,
                type: 'get',
                data:{"timed": new Date().getTime()},
                success: function(ret, textStatus, jqXHR) {
                   $('#main').append('<p>' + ret + '</p>');
                   fetch();
                },
                error: function(jqXHR, textStatus, errorThrown) {
                   setTimeout(fetch, 5000);
                },
                timeout:4000
             });
          }
          $(function(){
             fetch();
          });
      </script>
     </head>
 <body>
    <div id="main"></div>
 </body>
 </html>
```

服务器端的代码为：

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
          throws IOException, ServletException
      {
          long timed = Long.parseLong(req.getParameter("timed"));
          PrintWriter writer = res.getWriter();
          Random rand = new Random();
          try {
             // 循环 查询有无数据变化
             while (true) {
                Thread.sleep(200); // 休眠 300 毫秒，模拟处理业务等
                int i = rand.nextInt(100); // 产生一个 0-100 之间的随机数
                if (i > 20 && i < 55) {
                    // 如果随机数在 20-55 之间就视为有效数据，模拟数据发生变化
                   long responseTime = System.currentTimeMillis();
                   // 返回数据信息，请求时间、返回数据时间、耗时
                   writer.print("{result: " + i + ", response time: " +
                     responseTime + ", request time: " + timed + ", use
                     time:" + (responseTime - timed)+"}");
                   break; // 跳出循环，返回数据
                } else { // 模拟没有数据变化，将休眠保持连接
                   Thread.sleep(1300);
                }
             }
```

```
        } catch (Exception e) {}
    }
```

在服务器端模拟数据发生变化，并将数据“推送”到客户端，显示效果如图 7-8 所示。

```
{result: 30, response time: 1427346792627, request time: 1427346790710, use time: 1917}
{result: 42, response time: 1427346792940, request time: 1427346792630, use time: 310}
{result: 44, response time: 1427346793251, request time: 1427346792941, use time: 310}
{result: 37, response time: 1427346802566, request time: 1427346802258, use time: 308}
{result: 23, response time: 1427346811883, request time: 1427346811570, use time: 313}
{result: 48, response time: 1427346822801, request time: 1427346820888, use time: 1913}
{result: 21, response time: 1427346832135, request time: 1427346831807, use time: 328}
```

图 7-8　长轮询示例的显示效果

本章小结

本章首先介绍了 Ajax 的原理。Ajax 不是一种技术而是几种原有技术的组合。Ajax 技术比传统的 Web 应用有许多优势，但也存在安全问题与客户端代码过于复杂等问题。然后讲解了$.get()、$.getJSON()等实用方法，如果需要最大的灵活性，那么可以使用带有丰富分类选项的$.ajax()方法。最后介绍了 jQuery 提供的 Ajax 全局事件和 Ajax 全局设置函数。在掌握了这些令人印象深刻的 Ajax 实用函数后，我们在 Web 中实现丰富的功能将会更容易。

本章练习

1. 什么是 Ajax？
2. 请介绍一下 XMLhttprequest 对象？
3. jQuery 中关于 Ajax 的方法有哪些？

第8章　jQuery插件

jQuery 的易扩展性吸引了大量开发者共同编写 jQuery 插件。jQuery 插件不仅能够增强网站的可用性，有效地改善用户体验，还可以大大减少开发时间。本章将重点介绍几个常用的 jQuery 插件。

本章内容：

- 如何创建 jQuery 插件
- 如何选择 jQuery 插件
- 掌握 jQuery Image Select 插件、jQuery templating 插件
- 掌握 jQuery UI 插件

8.1　jQuery 插件介绍

插件是实现了某种软件接口的程序模块，第三方开发者可以遵循该接口开发独立的功能。为了满足 Web 开发者不断增长的需求，基于 jQuery 的插件已经达到 2000 多个，jQuery 已经远不只是核心库。这些插件大都来自 jQuery 社区，现在 jQuery 社区已经成长为一个生态系统。

开发者可使用 jQuery 插件来完成表单确认、各类图表、字段提示、动画、进度条等任务，也有一些比较复杂的 jQuery 插件(比如 jQuery UI)，jQuery UI 是一个丰富的用户界面插件，提供了一些常见的事件、约定以及视觉样式。JQeury 插件不仅可以给我们解决很多问题，而且使用非常简单。

为了让其他人甚至自己重用自己编写的代码，我们会希望把这些代码打包成一个插件。下面就介绍两种创建插件的方法。

8.1.1　通过全局函数创建插件

jQuery 提供的一些功能是通过全局函数给出的，例如$.ajax()函数，$.ajax()所做的一切都可以通过简单地调用一个名为 ajax 的常规全局函数来实现，最简单的创建 jQuery 插件的方式就是在 jQuery 命名空间(或者理解成 jQuery 自身)上添加了一个全局方法。例如：

```
jQuery.sayHello = function() {
    console.log("hello");
}
```

使用的时候就如同调用 jQuery 的其他函数一样：

```
$.sayHello()
```

jQuery 提供的这个方法也可以用来扩展 jQuery 对象。如果我们想在插件中提供多个全局函数，使用$.extend()方法会更方便、更加清晰，$.extend()的语法如下：

```
jQuery.extend(deep, target, object1,object2,.. objectN)
```

表 8-1 给出了$.extend()方法的参数。

表 8-1　$.extend()方法的参数

参　数	描　述
deep	可选。如果设置为 true，则递归合并
target	可选。待修改对象
object1	待合并到 target 的对象

第 1 个参数 deep 表示是否进行深度拷贝，我们看一个例子：

```
var result=$.extend( true, {},
{ name: "John", location: {city: "Boston",county:"USA"} },
{ last: "Re", location: {state: "MA",county:"China"} } );
```

可以看出第 1 个深度拷贝参数为 true，第 3 个参数中嵌套子对象 location:{city:"Boston"}，第 4 个参数中也嵌套子对象 location:{state:"MA"}，那么合并后的结果就是：

```
result={name:"John",last:"Re",location:{city:"Boston",state:"MA",
    county:"China"}}
```

那么我们可以使用$.extend()方法扩展 jQuery 的功能，下面的代码演示了一个包含两个函数的插件：

```
jQuery.myPlugin = $.extend(
{
    sayHello: function() {
        console.log("hello");
    },
    sayHi : function() {
        alert("hi");
    }
});
```

我们创建了另一个命名空间 jQuery.myPlugin，使用时实际上调用的是 myPlugin 属性上的函数，例如：

```
$.myPlugin.sayHello();
$.myPlugin.sayHi();
```

8.1.2　通过$.fn()方法创建插件

使用全局函数的方法无法利用 jQuery 强大的选择器带来的便利，要处理 DOM 元素以及将插件更好地运用于所选择的元素身上，还需要使用第二种开发方式，例如：

```
$.fn.pluginName = function() {
   console.log("do something in plugin");
}
```

jQuery 中的大多数操作都需要通过选择器得到对象，然后再操作 DOM 元素。而在使用上面的方法创建的插件内部，关键字 this 引用的就是当前的 jQuery 对象。因而，可以在 this 上面调用任何内置的 jQuery 方法，例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
</head>
<body>
<ul>
    <li>
        <a href="http://www.webo.com/">我的微博</a>
    </li>
    <li>
        <a href="http://www.cnblogs.com/">我的博客</a>
    </li>
    <li>
        <a href="http://baidu.com/">我的小站</a>
    </li>
</ul>
<p>这是 p 标签不是 a 标签，我不会受影响</p>
<script>
    $.fn.myLink = function() {
        //在这里面,this 指的是用 jQuery 选中的元素
        this.css('color', 'red');
    };
    $(function(){
        $('a').myLink();
    })
</script>
</body>
<html>
```

$('a')返回的是页面上所有 a 标签的集合，且这个集合已经是 jQuery 包装类型了，也就是说，在对其进行操作的时候可以直接调用 jQuery 的其他方法而不需要再用美元符号来包装。调用$('a').myLink()后，页面上所有链接的颜色会变成红色。

如果需要对一个集合进行处理，我们可以针对每个元素进行相应操作。比如现在要在每

个链接处显示链接的真实地址，首先通过 each 遍历所有 a 标签，然后获取 href 属性的值再加到链接文本后面。代码如下：

```
<script>
    $.fn.myLink = function() {
   //在这里面,this 指的是用 jQuery 选中的元素
   this.css('color', 'red');
   this.each(function() {
       //对每个元素进行操作
       $(this).append(' ' + $(this).attr('href'));
   })};

    $(function(){
         $('a').myLink();
    })
</script>
```

我们已经知道 this 指代 jQuery 选择器返回的集合，那么通过调用 jQuery 的.each()方法就可以处理集合中的每个元素了，但此刻要注意的是，在 each 方法内部，this 指向的是普通的 DOM 元素，如果需要调用 jQuery 的方法，就需要用$来重新包装。上面的代码的运行效果如图 8-1 所示：

- 我的微博 http://www.webo.com/
- 我的博客 http://www.cnblogs.com/
- 我的小站 http://baidu.com/

这是p标签不是a标签，我不会受影响

图 8-1　通过$.fn()方法创建插件

8.1.3　让插件接收参数

稍微复杂一些的插件，函数的执行可能依赖于用户的输入。我们把这些用户输入的值定义为参数，比如现在不想让链接只变成红色，我们让插件的使用者自己定义显示什么颜色，要做到这一点很方便，只需要使用者在调用的时候传入一个参数即可。代码如下：

```
<script>
    $.fn.myLink = function(color) {
    //在此处,this 指的是用 jQuery 选中的元素
    this.css('color', color);
    this.each(function() {
       //对每个元素进行操作
       $(this).append(' ' + $(this).attr('href'));
    })};
    $(function(){
         $('a').myLink("#00FF00");
    })
</script>
```

随着参数的增多，参数的次序会让使用者望而却步，而且指定的每个参数并不是必不可

少的。解决办法就是只在使用者调用的时候传入一个对象参数即可，所有传入的参数都设置为这个传入对象的属性。例如：

```
$.fn.myPlugin = function(options) {
    var defaults = {
        'color': 'red',
        'fontSize': '12px'
    };
    var settings = $.extend(defaults, options);
    return this.css({
        'color': settings.color,
        'fontSize': settings.fontSize
    });
}
```

同时为了灵活，使用者可以不传递参数，上面的代码里面会给出参数的默认值，如果对象 options 中有同名属性时，合并的时候 options 对象会覆盖前面的 defaults 对象。这样就实现了用户指定了值的参数使用指定的值，未指定值的参数使用插件的默认值。上面的例子中还指定了另一个参数 fontSize，允许使用插件的时候设置字体大小。

现在，如果我们调用插件的时候指定颜色，字体大小未指定，则会使用插件里的默认值 12px。代码如下：

```
$('a').myPlugin({
    'color': '#2C9929'
});
```

我们知道调用 extend 时会将 defaults 的值改变，这样会破坏插件的一致性，因为它作为插件有一些东西应该保持一致，另外就是如果在后续代码中还要使用这些默认值的话，当我们再次访问它时它已经被用户传进来的参数更改了。那么我们可以将一个新的空对象作为 $.extend 方法的第一个参数，defaults 和用户传递的参数对象紧随其后，这样做的好处是所有值被合并到这个空对象上，保护了插件里面的默认值。代码如下：

```
var settings = $.extend({},defaults, options);//将一个空对象作为第一个参数
```

通过全局函数创建的插件类似于在 jQuery 对象上添加了一个全局方法，是类级别的。而通过$.fn()方式创建的插件是对象级别的，下面是 jQuery.fn 的源代码：

```
jQuery.fn = jQuery.prototype = {
//代码
}
```

可以清晰地看出 jQuery.fn 实际上是 jQuery 的原型对象。而根据 JavaScript 原型的知识，jQuery.prototype 原型对象上的属性、方法和事件，将扩展到所有基于该原型的实例对象。这就是我们建议使用$.fn()方式创建插件的原因。

到这里，我们已经可以开发一个自己的 jQuery 插件了，下面给出一些开发插件的原则：

- 为插件设置一个适当的命名空间是插件开发中非常重要的一部分。合理的命名空间

可以降低插件被另一些插件或者当前页面上的代码覆盖的几率。命名空间也可以让开发工作变得容易，它可以让我们更好地跟踪方法、事件和数据。

- 用自调用匿名函数包裹我们的代码。函数可以形成一个作用域，域内的代码是无法被外界访问的。如果我们将自己的代码放入一个函数中，那么就不会污染全局命名空间，同时也不会和别的代码冲突。代码如下：

```
(function() {
     $.fn.myLink = function(color) {
    //在这里面,this 指的是用 jQuery 选中的元素
    this.css('color', color);
    this.each(function() {
        //对每个元素进行操作
        $(this).append(' ' + $(this).attr('href'));
    })};
})();
```

- 一个单独的插件在 jQuery.fn 对象上的命名空间都不应超过一个。
- 不要为插件函数定义过多的参数，而是通过传递一个可被扩展的对象字面量来传递参数。

8.2　寻找插件

当我们开始 jQuery 开发项目时，还要决定是使用插件来构建程序，还是什么都自己写。但大多数情况下，使用其他开发人员发布的开源插件会更好。一方面是插件背后有庞大的社区支持，另一方面是会有大量的用户发现其中的 bug。

8.2.1　搜索 jQuery 插件

jQuery 提供了官方的插件网站 http://plugins.jQuery.com/，里面囊括了大量插件，并给出了每个插件的用户评级、版本及 bug 报告，如图 8-2 所示。



图 8-2　jQuery 官网插件

通常一个插件的品质越高，它的用户评级也越高。其中每个插件都提供 zip 文件下载、示例代码和教程的链接。当然还有其他没有发布到 http://plugins.jQuery.com/上的插件，只能通过搜索引擎或者 https://github.com/来查找我们想要的插件。

8.2.2　选择 jQuery 插件

寻找到好用的 jQuery 插件并不容易，因为很多插件在工程化上做得比较差，缺乏足够的灵活性，难以融入我们的项目。那么如何评估一个插件是否是品质优良的插件呢？如果功能非常简单，我们应该自己亲手写一个，如果插件不能满足我们的需求，我们可以花点精力扩展那个插件。插件的学习难度如何，首先要看插件的文档，好的插件会有详细的说明文档和示例教程。看看插件最近的更新情况，是否有开发人员持续稳定地对其进行维护。检查插件是否对 HTML 标记有依赖，一些奇怪或严格的 HTML 标记会让插件非常难用，因为我们需要调整 HTML 标记以适应插件的具体要求。如果插件中的 CSS 品质不好，那也表明开发者在前端技能上有所欠缺。如果插件提供了单元测试，那最好不过了，这样我们可以在希望的环境中运行一下单元测试，有单元测试的插件表明插件的质量比较好。最后还应该了解下有多个人在使用这个插件，用户的评级高肯定是个不错的现象。

在人人微博网站，上传个人头像功能中需要对图片进行剪裁以生成大小为 100*100 的头像缩略图，我们使用 jQuery Image Select 插件来实现。在微博列表功能中，每一篇微博都会使用相同的 HTML 和 CSS 模板，我们使用 jQuery templating 插件来实现。

8.3　jQuery ImageArea Select 插件介绍

8.3.1　jQuery ImageArea Select 下载及安装

jQuery ImageArea Select 是一个允许用户使用简单、直观的点击、拖动界面图像选择矩形区域的 jQuery 插件。该插件可用于 Web 应用程序中轻松实现图像裁剪功能、照片标记功能、图像编辑功能等。官方网站为 http://odyniec.net/projects/imgareaselect/，官网上提供了在线示例和对各个浏览器的支持情况，如图 8-3 所示。

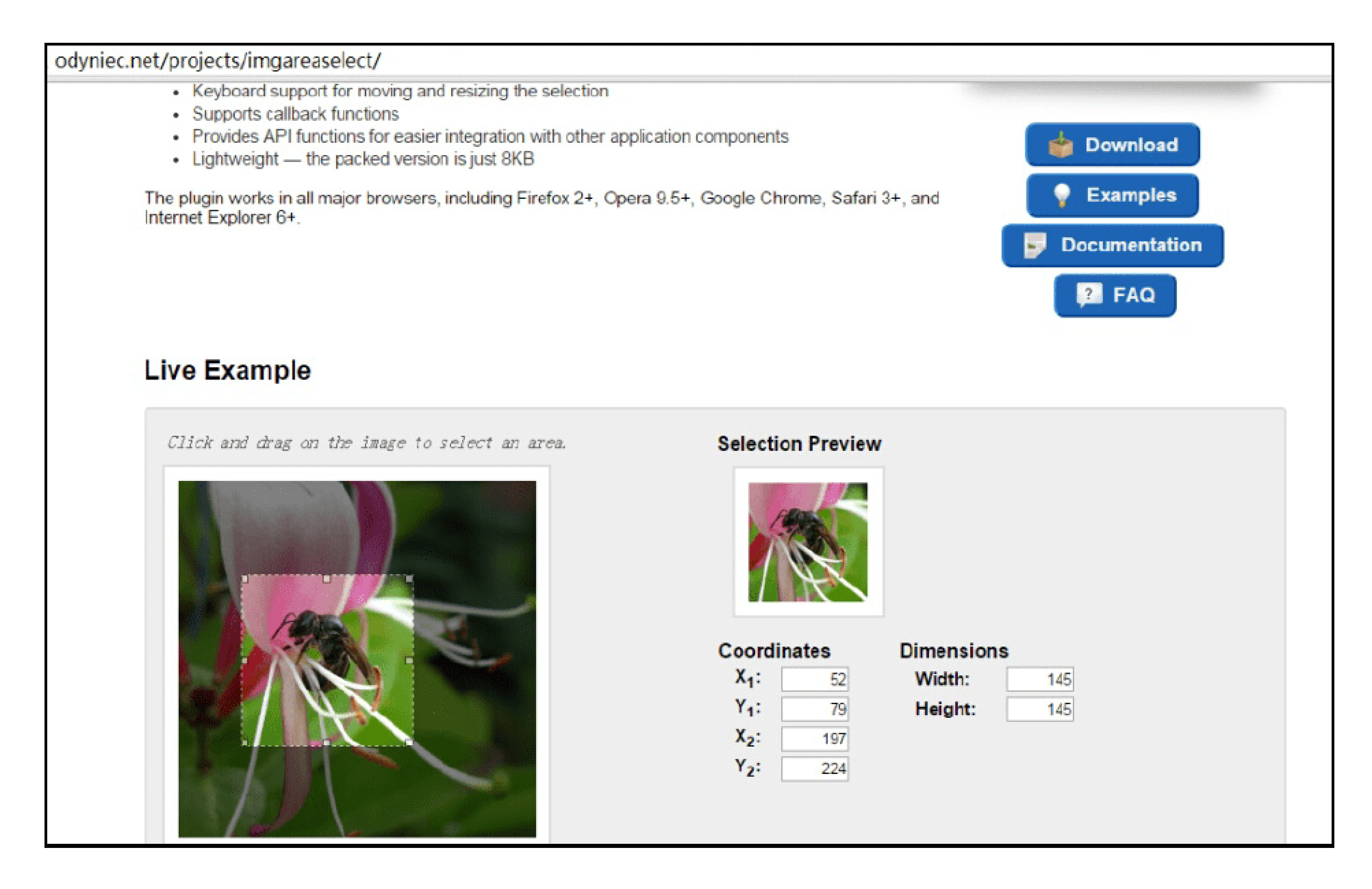


图 8-3　jQuery ImageArea Select 插件

8.3.2　使用 jQuery ImageArea Select

我们可以在$(document).ready() 或 $(window).load() 事件处理程序中初始化插件。例如：

```
$(document).ready(function () {
$('img#photo').imgAreaSelect({
handles: true,
onSelectEnd: someFunction
});
});
```

传递给插件的可选参数对象如表 8-2 所示。

表 8-2　imgAreaSelect 方法的参数

参 数 名	数据类型	默 认 值	描　　述
aspectRatio	string		设定选取区域的显示比率，如 4:3
autoHide	boolean	false	如果设置为 true，则当选择区域选择结束时消失
classPrefix	string	imgareaselect	这是一个字符串，表示插件样式的类名加前缀
disable	boolean		如果设置为 true，禁用插件
enable	boolean		如果设置为 true，插件被重新启用
fadeSpeed	boolean	false	如果设置为大于零的数字，则用优美的淡入/淡出动画来显示图片
handles	boolean	false	如果设置为 true，调整手柄则会显示在选择区域内，如果设置为 corners，则只有角处理会显示调整手柄
hide	boolean		如果设置为 true，选择范围是隐藏

(续表)

参数名	数据类型	默认值	描　述
imageHeight	number		图片的真实高度
imageWidth	number		图片的真实宽度
instance	number		如果设置为 true，imgAreaSelect()调用返回一个 imgAreaSelect 绑定到的图像的实例，使我们可以使用它的 API 方法
keys	boolean	false	启用/禁用键盘支持，默认值为 false
maxHeight	number		选取的最大高度(单位为像素)
maxWidth	number		选取的最大宽度(单位为像素)
minHeight	number		选取的最小高度(单位为像素)
minWidth	number		选取的最小宽度(单位为像素)
movable	boolean	true	确定选取是否可以移动
parent	string	body	一个 jQuery 对象或选择字符串，指定被追加到父元素
persistent	boolean	false	如果设置为 true，选择区域以外的点击将不会启动一个新的选区(即用户只能移动/调整现有的选择范围)，默认值为 false
resizable	boolean	true	确定是否选择面积应可调整大小
show	boolean		如果设置为 true，选区则会显示
x1 y1			最初选择区域的左上角坐标
x2 y2			最初选择区域的右上角坐标

一般情况下，在选区选择完毕后，根据选择的大小和位置处理成我们想得到的结果就可以了。提供的 onSelectEnd 事件函数就是最常用的回调函数，例如：

```
$('img#photo').imgAreaSelect({
        onSelectEnd: function (img, selection) {
        alert('width: ' + selection.width + '; height: ' + selection.height);
        }
});
```

第一个参数是插件所附加到的图像的引用，第二参数是表示当前选定内容的对象，该对象有 6 个属性：x1、y1 表示所选区域的左上角的坐标；x2、y2 表示所选定区域右下角的坐标；width 表示选择宽度；height 表示选择高度。除了 onSelectEnd 回调函数，还有 onInit、onSelectStart、onSelectChange 回调函数，它们都具有相同的参数。

下面我们来实现头像缩略图功能，将使用 jQuery ImageArea Select 插件选择区域图并上传到服务端。在第 2 章最后我们完成了个人信息页面的表单设计，在此基础上完成头像缩略图功能。

HTML5 代码为：

```
<div id="midleft2">头像：</div>
    <div id="uploader">
    <form name="" id="uploadForm" method="post" enctype="multipart/form-data"  >
    <div>
          <input name="photo" id="file"  size="27" type="file" multiple=
            "multiple"  accept="image/gif, image/jpeg,image/x-png" form=
            "uploadForm"/>
          <input id = "upload_but" name="upload_but" value="上传" type= "submit"
              form="uploadForm"/>
          <input type="hidden" id="y1" class="y1" name="y1" form="uploadForm"/>
          <input type="hidden" id="x1" class="x1" name="x1" form="uploadForm"/>
          <input type="hidden" id="y2" class="y2" name="y2" form="uploadForm"/>
          <input type="hidden" id="x2" class="x2" name="x2" form="uploadForm"/>
     </div>
     </br>
     <div id="div_big_view" >
          <img id="big_view" src="" form="infoForm"/>
     </div>
     </form>
     </div>
...................
     <!-- 缩略图区域  -->
     <div id="div_preview" >
      <img id="crop_preview" src="" form="infoForm"/>
     </div>
```

JavaScript 代码为：

```
//建立一个可存取到该 file 的 url
function getObjectURL(file) {
    var url = null ;
    if (window.createObjectURL!=undefined) { // basic
        url = window.createObjectURL(file) ;
    } else if (window.URL!=undefined) { // mozilla(firefox)
        url = window.URL.createObjectURL(file) ;
    } else if (window.webkitURL!=undefined) { // webkit or chrome
        url = window.webkitURL.createObjectURL(file) ;
    }
    return url ;
}
//选择图片区域后回调函数
function preview(img, selection) {
    if (!selection.width || !selection.height)
        return;
    //缩略图的大小为 100px
    var scaleX = 100 / selection.width;
    var scaleY = 100 / selection.height;

    $('#crop_preview').css({
```

```
            //大图大小为 300px
            width: Math.round(scaleX * 300),
            height: Math.round(scaleY * 300),
            marginLeft: -Math.round(scaleX * selection.x1),
            marginTop: -Math.round(scaleY * selection.y1)
        });
        //设置数据值
        $('#x1').val(selection.x1);
        $('#y1').val(selection.y1);
        $('#x2').val(selection.x2);
        $('#y2').val(selection.y2);
    }
    $(document).ready(function() {
         //选择图片后触发
        $("#file").change(function(){
            var objUrl = getObjectURL(this.files[0]) ;
            console.log("objUrl = "+objUrl) ;
            if (objUrl) {
                $("#big_view").attr("src", objUrl) ;
                $("#crop_preview").attr("src", objUrl) ;
            }
        }) ;
         //使用插件并绑定回调函数
         $('#big_view').imgAreaSelect({ aspectRatio: '1:1', handles: true,
            fadeSpeed: 200, onSelectChange: preview });
    });
```

现在页面效果就做好了，上传图片后，可以将选区以 100*100 像素大小的缩略图显示。最后还应该把缩略图上传到服务器保存起来。服务器端提供的上传图片的 API 为：上传头像缩略图功能 API。

上传头像缩略图功能 API

接口说明：上传头像缩略图

URL：http://localhost:8080/rrwb/crud/uploadServlet

HTTP 请求方式：POST/GET

enctype：multipart/form-data

请求参数如表 8-3 所示。

表 8-3　请求参数

参数名称	类　型	说　明
x1 y1	Number	最初选择区域的左上角坐标
x2 y2	String	最初选择区域的右上角坐标

上传图片的 form 表单中，enctype 属性设置为"multipart/form-data"，以二进制格式上传

图片。

响应参数如表 8-4 所示。

表 8-4　响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
url	Number	返回的头像缩略图的 url 地址

实现上传头像缩略图的 JavaScript 代码为：

```
$("#uploadForm").submit(function(e) {
   e.preventDefault();
   if(window.FormData !== undefined)  // for HTML5 browsers
   {
       var formData = new FormData(this);
       $.ajax({
           url: 'http://localhost:8080/rrwb/crud/uploadServlet',
           type: 'POST',
           mimeType:"multipart/form-data",
           data:formData,
           contentType: false,
           cache: false,
           processData:false
       })
       .done(function(data) {
       console.log(data);
       var response = $.parseJSON(data);
           if(response.code == 1){
            $("#crop_preview").removeAttr("style")
            $("#crop_preview").attr("src", response.url) ;
            alert("上传成功！");
           }
          })
       }
       else  //for olden browsers
       {
        alert('浏览器不支持 HTML5');
       }
});
```

代码中使用了 FormData 对象，FormData 对象提供了一种方法来轻松地构建一组键/值对，表示表单字段及其值，除了普通的字符串类型，它的值还可以是 Blob 对象或 File 对象，然后可以使用 XMLHttpRequest 的 send()方法以“multipart/form-data”的格式发送数据。使用 FormData 对象，就可以在异步请求中发送选择的文件。

8.4　jQuery templating 插件介绍

8.4.1　jQuery templating 下载及安装

随着前端交互的复杂性不变提升，无刷新页面数据传输与渲染越发地频繁，我们发现传统的前端开发方式在 Ajax 数据渲染等方面存在一个主要问题：繁琐的数据渲染。当前端从后台通过 Ajax 等方式获取数据后，如果要将这个数据渲染到指定的 dom 元素中，则需要进行各种字符串拼接、字符串替换工作。例如人人微博网站中根据昵称查找好友的功能中，前端需要循环处理 Ajax 方式获取的数据，并创建许多类似的元素插入到 DOM 中，当访问 http://localhost:8080/rrwb/guangchang.html?_id=16&nick_name=zhang 时，效果如图 8-4 所示。



图 8-4　好友广场

HTML5 代码如下：

```
<div id="main">
    <!--广场标题-->
    <div id="more">
    <img src="images/img44.png" />
    </div>
    <!--/广场标题-->
    <!--好友列表-->
    <div id="content">
    </div>
    <!--好友列表结束-->
</div>
```

JavaScript 代码如下：

```
<script type="text/javascript">
  var template = '<div id="content"><div id="tu2"><a href="friend.html?
  fans_id='+getWindowParam("_id")+'&followed_id={_id}">{nick_name}<br />
  <img src="{img}" border="0"></a></div> ';
  $(document).ready(function() {
      $.ajax({
     url: 'http://localhost:8080/rrwb/crud/user/findUserLikeNickName',
```

```
        type: 'POST',
        dataType: 'json',
       data:{nick_name:getWindowParam("nick_name"),my_id:getWindowParam("_id")}
      })
      .done(function(response) {
        if(response.code == "1"){
          if(response.data.length>0){
            $.each(response.data,function(i,item){
                    var str = substitute(template,item);
                    $(str).appendTo($('#content'));
                   }
                );
          }
        }
      })
    });
  /**
   *获取地址栏参数
   */
  function getWindowParam(name){
     if(name=(new RegExp('[?&]'+encodeURIComponent(name)+'=([^&]*)')).exec
        (location.search))
        return decodeURIComponent(name[1]);
  }
  /**
   * 替换字符串中的字段
   * @param {String} str 模板字符串
   * @param {Object} o json data
   * @param {RegExp} [regexp] 匹配字符串的正则表达式
   */
  function substitute(str,o,regexp){
    return  str.replace(regexp || /\{([^{}]+)\}/g, function (match, name) {
      return (o[name] === undefined) ? '' : o[name];
    });
  }
  </script>
```

这里使用字符串拼接创建了一个模板，在得到数据后使用 substitute 函数替换大括号中的动态数据。这种方式是繁琐且费时的，在需要修改模板时，都需要重新花费不少时间与精力。那有什么方法可以解决这个问题呢？于是就出现了 js 模板，js 模板引擎越来越多地得到应用，如今已经出现了几十种 js 模板引擎，国内各大互联网公司也都开发了自己的 js 模板引擎。模板的工作原理可以简单地分成两个步骤：模板解析(翻译)和数据渲染。当我们在 JSP 中写 <%=name%> 的时候，其实就是在应用模板，在后台这句话会被转换成 out.print("
"+name+"
")。模板的数据渲染就是把模板中的占位符(这里是 name)替换成传入的值。有的模板是依赖于 jQuery，而有的模板是独立于 jQuery 的。下面介绍 jQuery

的一个模板插件——jQuery templating。

jQuery templating 是一个旨在简化模板操作的 jQuery 插件，它支持加载外部的 HTML 模板或者使用 jQuery 模板对象。它的语法简洁明了，使用 HTML5 的自定义属性绑定数据，还能够自动把数据转化为自己希望的格式。例如下面的模板：

```
<script type="text/html" id="template">
      <div data-content="author"></div>
      <div data-content="date"></div>
      <img data-src="authorPicture" data-alt="author"/>
      <div data-content="post"></div>
   </script>
$("#template-container").loadTemplate(
        $("#template"),
        {
          author: 'Joe Bloggs',
          date: '25th May 2013',
          authorPicture: 'Authors/JoeBloggs.jpg',
          post: 'This is the contents of my post'
        }
        );
```

上面的代码在 script 标签中定义了一个模板，并指定 id 为 template。在目标元素 template-container 上使用模板和数据，这样就可以渲染模板并插入到目标元素中。运行效果如图 8-5 所示。

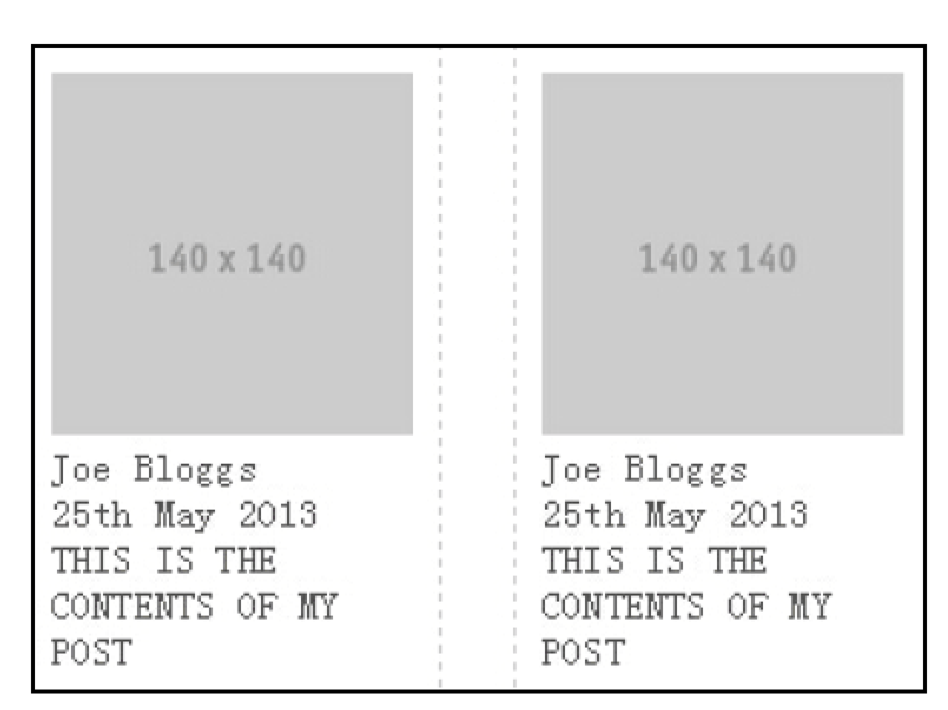


图 8-5　loadTemplate 方法

我们还可以把模板保存在外部的 HTML 文件中，使用方法如下：

```
$("#template-container").loadTemplate(
    "Templates/template.html",
    {
      author: 'Joe Bloggs',
      date: '25th May 2013',
      authorPicture: 'Authors/JoeBloggs.jpg',
      post: 'This is the contents of my post'
    }
);
```

不同的元素通过“data-*”属性绑定不同的数据，如表 8-5 所示。

表 8-5　jQuery templating 绑定数据参数

属　　性	说　　明
data-content	绑定数据到元素的内容，等价于$(elem).html(value)
data-content-append	追加数据到元素内容的最后位置，等价于$(elem).append(value)
data-content-prepend	追加数据到元素内容的开始位置，等价于$(elem).prepend(value)
data-id	设置元素的 id 值，等价于$(elem).attr("id", value)
data-href	等价于$(elem).attr("href", value)
data-value	$(elem).val(value)
data-link	把给定的值包装为链接
data-options	设置下拉选择框的选项，要求值为字符串数组
data-format	设置数据格式处理器

使用数据格式处理器之前需要先添加数据格式处理函数，代码如下：

```
$.addTemplateFormatter({
       UpperCaseFormatter : function(value, template) {
              return value.toUpperCase();
          },
       LowerCaseFormatter : function(value, template) {
              return value.toLowerCase();
          },
       SameCaseFormatter : function(value, template) {
              if(template == "upper") {
                     return value.toUpperCase();
                 } else {
                     return value.toLowerCase();
                 }
          }
   });
```

绑定数据时，指定数据格式处理属性的代码如下

```
<div data-content="" data-format="SameCaseFormatter" data-format-options=
     "upper"></div>
```

loadTemplate 还可以通过第三个参数(options)指定自定义的选项，见表 8-6。

表 8-6　loadTemplate 方法的 options 参数

参　　数	说　　明
overwriteCache	默认为 false。是否重新加载模板
success	成功完成后的回调函数
error	错误处理回调函数
paged	默认值 false
pageNo	指定当前显示第几页

(续表)

参　　数	说　　明
elemPerPage	每页的行数
append	默认为 false。是否追加到元素内容的尾部
prepend	默认为 false。是否追加到元素内容的头部

8.4.2　使用 jQuery templating

在人人微博网站中，微博列表是根据服务器端获取的 JSON 数据动态生成的，界面需求如图 8-6 所示。



图 8-6　微博列表

实现代码如下：

```
<!-- Template -->
<script type="text/html" id="template">
    <div class="cont_down41">
     <div class="cont_down4_a"><img  data-src="user_head_img"
       class="picture1" /></div>
     <div class="cont_down4_b">
       <span class="fo" data-content-prepend="user_nickName">: </span>
       <span data-content="content"></span>
       <br/>
       <div class="photoBox">
         <div class="loadingBox">
             <span class="loading"></span>
         </div>
       <img data-src="img" class="zoom" onclick="zoom_image($(this).parent());"/>
       </div>
```

```
            <div class="photoArea" style="display:none;">
             <img src="" class="minifier" onclick="zoom_image($(this).parent());" />
            </div>
          <div>
           <time data-content="timestamp" data-format="DateTimeFormatter"></time>
        </div>

    </script>
    <script>
      $.addTemplateFormatter({
            DateTimeFormatter : function(value, template) {
            var cTime = new Date(value)
          return cTime.getFullYear()+"-"+(cTime.getMonth()+1)+"-"+cTime.getDate()
            +" "+cTime.getHours()+":"+cTime.getMinutes()+":"+cTime.getSeconds();
        }
    });
         //Ajax 访问 http://localhost:8080/rrwb/crud/weibo/previous_weibos 返回的数据
         //这里使用模拟数据
         var weibos =JSON.parse('[{"_id":9,"img":"upload/1433464756875_small.jpg",
             "user_id":1,"user_nickName":"111111","user_head_img":"upload/
             1432713847988_cut_.jpg","content":"ggg","status":true,"timestamp":
             1433464761078},{"_id":8,"img":"upload/1433464739733_small.jpg","
             user_id":1,"user_nickName":"111111","user_head_img":"upload/
             1432713847988_cut_.jpg","content":"ffff","status":true,"timestamp":
             1433464748007},{"_id":7,"img":"upload/1433464557701_small.jpg",
             "user_id":1,"user_nickName":"111111","user_head_img":"upload/
             1432713847988_cut_.jpg","content":"eeeee","status":true,"timestamp":
             1433464564031}]');

          //添加到列表后面
          $("#weibos").loadTemplate("#template", weibos,{append:true});
    </script>
```

上面的代码将 HTML 标记与 JavaScript 分离了，为此我们把模板放在单独的<script>标签里，代码中粗体显示的是需要绑定的属性。需要注意 script 标签的 type 属性不是 text/javascript，而是用 text/html(还可以使用 text/template)表明这是个 HTML 模板，而不是 JavaScript。这一设置会阻止浏览器把它当成 JavaScript 运行其中的内容。

前端模板可以分为两大类：带有嵌入式 JavaScript 的模板和较少逻辑控制的模板。只能传入变量，并使用几个预先定义好的函数，属于较少逻辑控制的模板。带有嵌入式 JavaScript 的模板中常用的有 Underscore 模板。Underscore.js 是一个很精干的库，压缩后只有 4KB。它提供了几十种函数式编程的方法，弥补了标准库的不足，大大方便了 JavaScript 的编程。Underscore.js 定义了一个下划线(_)对象，函数库的所有方法都属于这个对象。这些方法大致上可以分成：集合(collection)、数组(array)、函数(function)、对象(object)和工具(utility)等 5 大类。其官方网站为 http://underscorejs.org/，如图 8-7 所示。

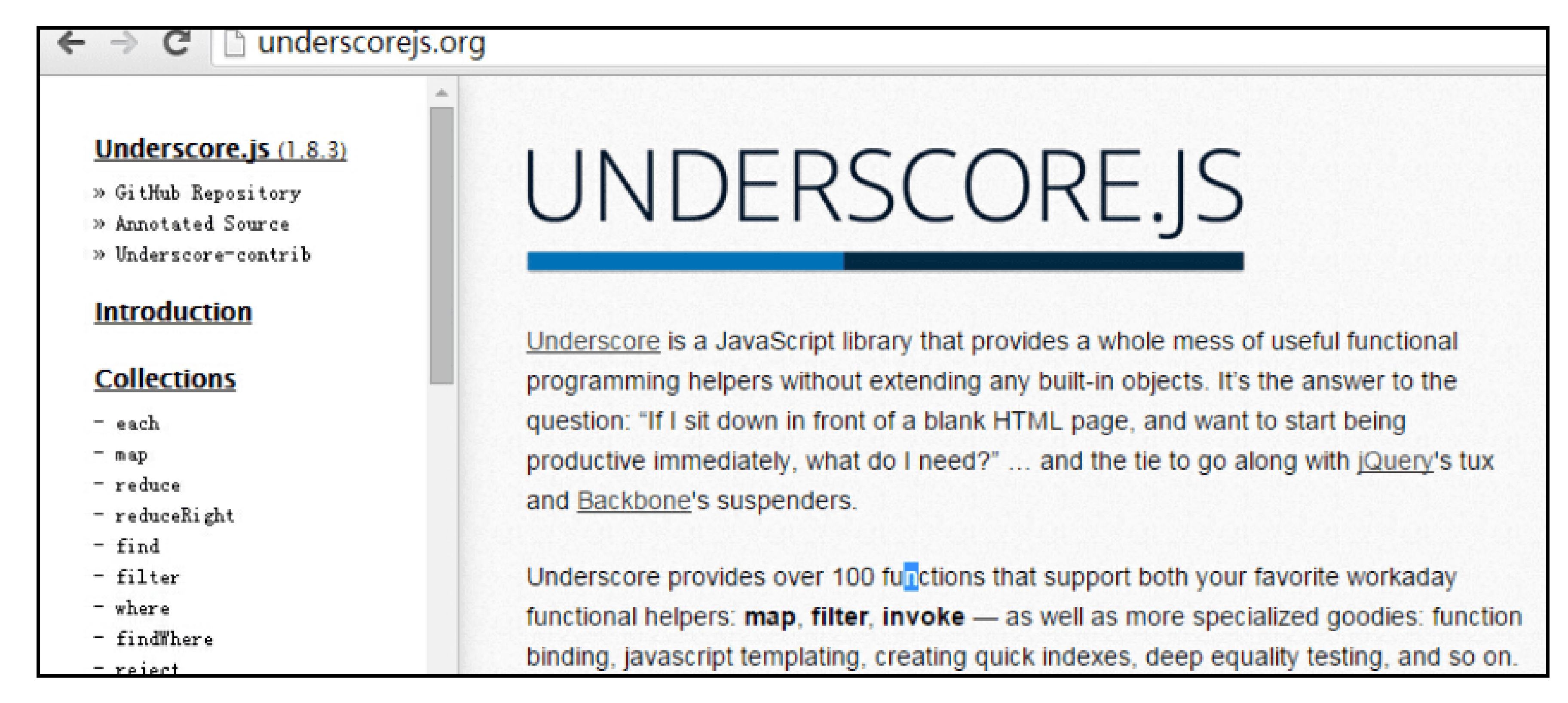


图 8-7　Underscore 官网

下面介绍 Underscore 模板的基础知识。第一步是使用_.template 方法创建一个模板：

```
var tmpl = _.template('<div>你好, <%= name%></div>');
```

第二步是编译模板。调用模板定义的函数并传入一个包含模板中定义的变量名称的属性的对象。

```
var result = tmpl ({name:'json'})
```

调用 console.log(result)会输出：

```
<div>你好, json</div>
```

我们还可以设置自己的界限符，代码如下：

```
_.templateSettings = {interpolate : /\{\{(.+?)\}\}/g};
var tmplN = _.template('<div>你好, {{name}}</div>');
var result = tmplN ({name:'json'});
console.log(result);
```

像使用 jQuery templating 一样，也可以把模板分离出来放在单独的 script 标签中，例如：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title></title>
  <script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
  <script src="http://underscorejs.org/underscore-min.js"></script>
</head>
<body>
</body>
<script type="text/template" id="my-template">
<article>
  <hgroup>
    <h1><%= title %></h1>
    <h2><%= subtitle %></h2>
  </hgroup>
```

```
  <p><%= description %></p>
</article>
</script>
<script>
var myTemplate = _.template( $('#my-template').text() );
var compiled = myTemplate({
  title: '使用Underscore模板',
  subtitle: '内部模板',
  description: '需要了解'
});
$('body').append(compiled);
</script>
</html>
```

使用内部模板还是外部模板各有优缺点。使用外部模板更具有灵活性，对于大型复杂页面更适合，同时也增加了管理不同页面模板的难度和工作量。一般会把模板放在页面内部，把模板放在页面内部的优点包括：把模板留在 HTML 中意义更明确，因为其他标记也在里面。它管理起来更容易。

Underscore 模板中还可以使用 JavaScript，这能让我们使用 JavaScript 控制渲染的结果。

基本的 if-else 语句很实用。比如微博列表模板中如果表示图片的变量是空字符串，就可以不显示图片的缩略图区域。代码如下所示：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
  <script src="http://underscorejs.org/underscore-min.js"></script>
</head>
<body>
</body>
<script type="text/template" id="my-template">
<div>
  <div> <%= name %><div>
  <% if (typeof img !== 'undefined' && img !=='')  { %>
         <img src="<%=img %>" />
  <% } else {%>
    没有图片
  <% }%>
</div>
</script>
<script>
var myTemplate = _.template( $('#my-template').text() );
var compiled = myTemplate({
  name: 'zhangsan',
  img: 'loading.gif'
});
$('body').append(compiled);
compiled = myTemplate({
```

```
  name: 'lisi'
});
$('body').append(compiled);
</script>
</html>
```

代码的运行效果如图 8-8 所示。

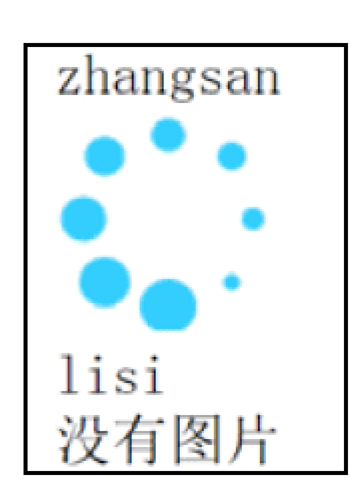


图 8-8　Underscore 中的 if-else

在组装列表时还可以使用循环语句，代码如下：

```
<h2><%= province %></h2>
<ul>
    <% for ( var i = 0, len = schools.length; i++ ) { %>
    <li><%= schools[i] %></li>
    <% } %>
</ul>
```

其实 Underscore.js 拥有自己特定的循环函数：_.each()。
使用 each 函数也同样可以完成组装列表的功能，并且更容易一些，代码如下：

```
<h2><%= province %></h2>
<ul>
    <% _.each(schools , function( item ) { %>
    <li><%= item %></li>
    <% } %>
</ul>
```

大型应用考虑更多的是响应速度，随着应用越来越复杂、模板越来越多，模板之间各种依赖出现的时候，我们的页面会变得臃肿不堪、难以维护，如何才是最佳的前端模板组织方案呢？前端模板预编译工具的出现解决了响应速度和管理模板的问题。它通过预编译技术让编译后的代码不依赖于模板引擎，异步加载预编译后的文件。它采用目录来组织维护前端模板，从而让前端模板实现工程化管理，最终保证前端模板在复杂单页 Web 应用下的可维护性。

8.5　jQuery UI 插件介绍

8.5.1　jQuery UI 下载及安装

jQuery UI 是一套 jQuery 的页面 UI 插件，包含很多种常用的页面控件，例如 Tabs、拉帘

效果、对话框、拖放效果、日期选择、颜色选择、数据排序、窗体大小调整等，它还包括许多 CSS 样式表。当我们访问 jQuery UI 网站(http://jQueryui.com/)时，可以快速下载通用版本，也可以自定义下载，如图 8-9 所示。

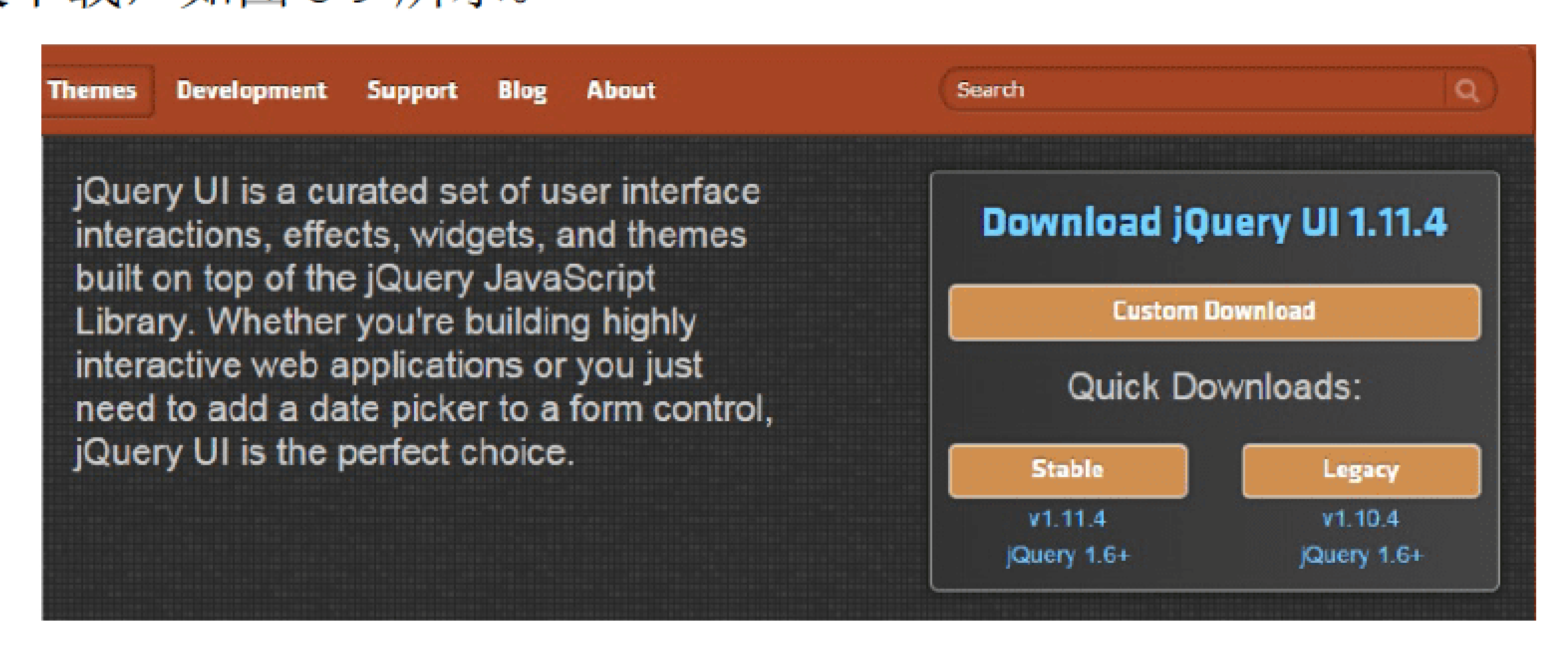


图 8-9　jQuery UI 官网

点击自定义下载的链接，我们可以组装适合自己的个性化包，可以去掉任何不需要使用的组件(例如 Datapicker 小部件)来缩小库的大小，如图 8-10 所示。

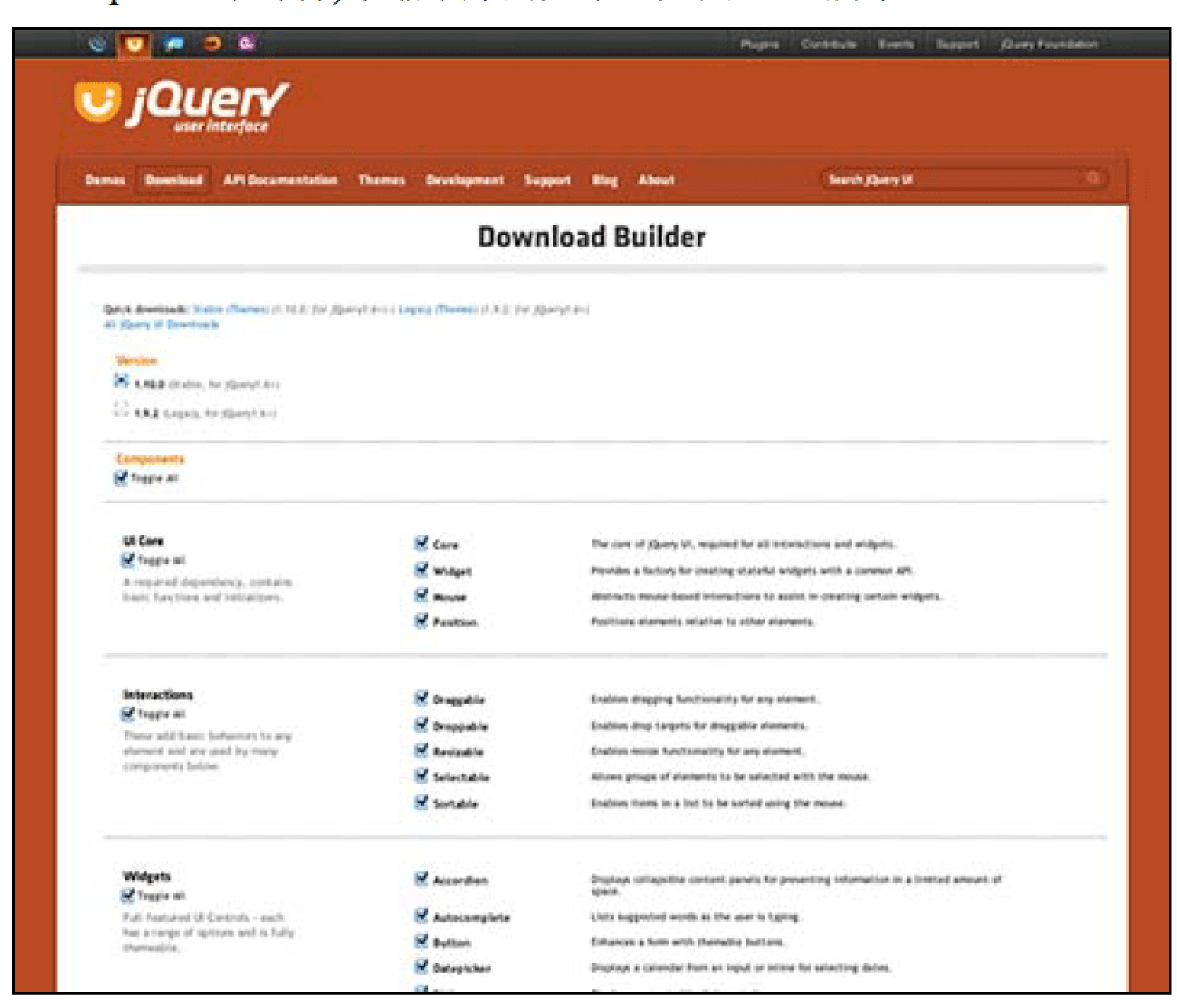


图 8-10　个性化包

下载生成器(Download Builder)页面的第一栏列出了 jQuery UI 所有的 JavaScript 组件分类：核心(UI Core)、交互部件(Interactions)、小部件(Widgets)和效果库(Effects)。jQuery UI 中的一些组件依赖于其他组件，当选中这些组件时，它所依赖的其他组件也都会自动被选中。我们所选的组件将会合并到一个 JavaScript 文件。

单击 Download 按钮，完成下载。我们将得到一个包含所选组件的自定义 zip 文件，解压后如图 8-11 所示。

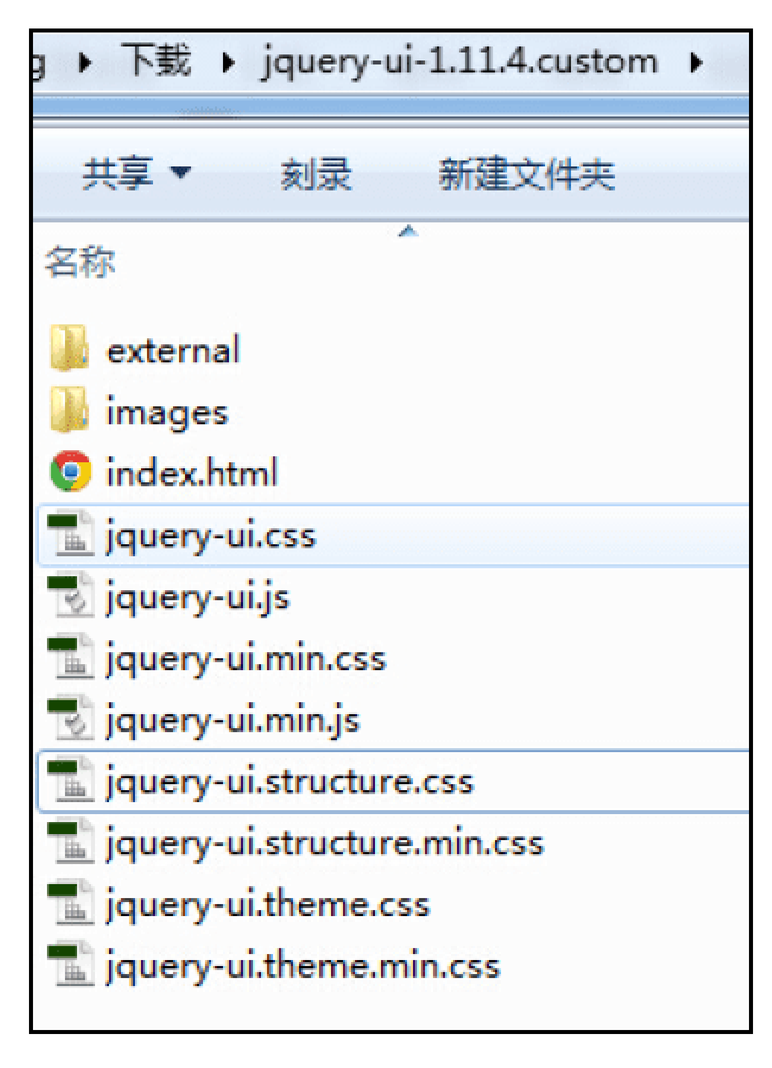


图 8-11　下载后的 jQuery UI

在文本编辑器中打开 index.html，index.html 提供了 jQuery UI 组件的演示例子，并且使用的是我们选择的样式。一般情况下，需要在 HTML 中引用三个文件，代码如下所示：

```
<link rel="stylesheet" href="jQuery-ui.min.css">
<script src="external/jQuery/jQuery.js"></script>
<script src="jQuery-ui.min.js"></script>
```

8.5.2　使用 jQuery UI

jQuery UI 能够对 HTML 元素进行增强，以增加与用户的交互。这些常见的交互行为包括 sortable、draggable、droppable、resizable 和 selectable。这也是 jQuery UI 的核心功能之一。

1. draggable 和 droppable

draggable 和 droppable 两者经常在一起使用，使我们能够单击并拖拽一个 HTML 元素，然后将其拖动到某个目标区域。代码如下：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI Droppable - Default functionality</title>
  <script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
  <link rel="stylesheet" href="jQuery-ui-1.11.4.custom/jQuery-ui.min.css">
  <script src="jQuery-ui-1.11.4.custom/jQuery-ui.min.js"></script>
  <style>
  #draggable { width: 100px; height: 100px; padding: 0.5em; float: left; margin:
     10px 10px 10px 0; }
  #droppable { width: 150px; height: 150px; padding: 0.5em; float: left; margin:
     10px; }
  </style>
  <script>
```

```
  $(function() {
    $( "#draggable" ).draggable();
    $( "#droppable" ).droppable({
      drop: function( event, ui ) {
        $( this )
          .addClass( "ui-state-highlight" )
          .find( "p" )
            .html( "Dropped!" );
      }
    });
  });
  </script>
</head>
<body>
<div id="draggable" class="ui-widget-content">
  <p>Drag me to my target</p>
</div>
<div id="droppable" class="ui-widget-header">
  <p>Drop here</p>
</div>
</body>
</html>
```

运行效果如图 8-12 所示。



图 8-12　jQuery UI 拖放

我们做的仅仅是在两个<div>元素上调用了 draggable 和 droppable 方法。在 droppable 方法中还使用了一个对象参数，它可以定义 droppable 行为的不同选项，我们使用了一个 drop 回调函数，用在某个元素释放于目标元素时触发。这种获取某个元素然后调用相关联的 jQuery UI 方法的模式是所有 jQuery UI 方法的基本模式。

2. resizable

resizable 行为能够使得 DOM 元素大小可以调整，它会为元素增加一个小握柄，并通过点击和拖动动作调整大小。代码如下：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI resizable - Default functionality</title>
```

```
  <script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
  <link rel="stylesheet" href="jQuery-ui-1.11.4.custom/jQuery-ui.min.css">
  <script src="jQuery-ui-1.11.4.custom/jQuery-ui.min.js"></script>
   <style>
  #resizable { width: 150px; height: 150px; background-color : #0099ff;  }
  #resizable h3 { text-align: center; margin: 0; }
  </style>
  <script>
  $(function() {
    $( "#resizable" ).resizable();
  });
  </script>
</head>
<body>
<div id="resizable" class="ui-widget-content">
  <h3 class="ui-widget-header">Resizable</h3>
</div>
</body>
</html>
```

运行效果如图 8-13 所示。

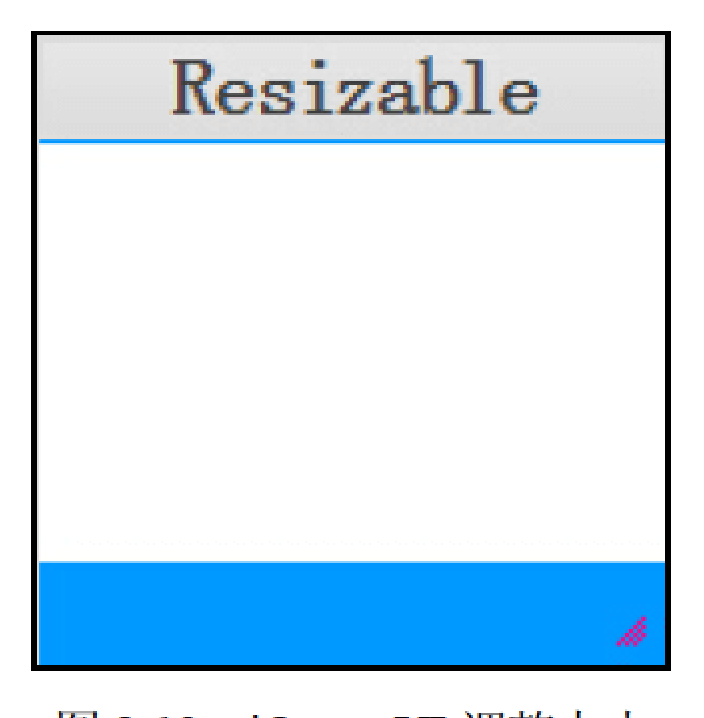


图 8-13　jQuery UI 调整大小

3. selectable

selectable 可以使得 DOM 元素可以被选择，通过这一功能可以用任意的 DOM 元素模拟复选框，点击时按住 Ctrl 键就可以实现多选。例如：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI resizable - Default functionality</title>
  <script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
  <link rel="stylesheet" href="jQuery-ui-1.11.4.custom/jQuery-ui.min.css">
  <script src="jQuery-ui-1.11.4.custom/jQuery-ui.min.js"></script>

</head>
<body>
 <style>
```

```
  #feedback { font-size: 1.4em; }
  #selectable .ui-selecting { background: #FECA40; }
  #selectable .ui-selected { background: #F39814; color: white; }
  #selectable { list-style-type: none; margin: 0; padding: 0; width: 60%; }
  #selectable li { margin: 3px; padding: 0.4em; font-size: 1.4em; height: 18px; }
  </style>
  <script>
  $(function() {
    $( "#selectable" ).selectable();
  });
  </script>
</body>
<ol id="selectable">
  <li class="ui-widget-content">可选择 1</li>
  <li class="ui-widget-content">可选择 2</li>
  <li class="ui-widget-content">可选择 3</li>
  <li class="ui-widget-content">可选择 4</li>
  <li class="ui-widget-content">可选择 5</li>
  <li class="ui-widget-content">可选择 6</li>
  <li class="ui-widget-content">可选择 7</li>
</ol>
</html>
```

运行效果如图 8-14 所示。

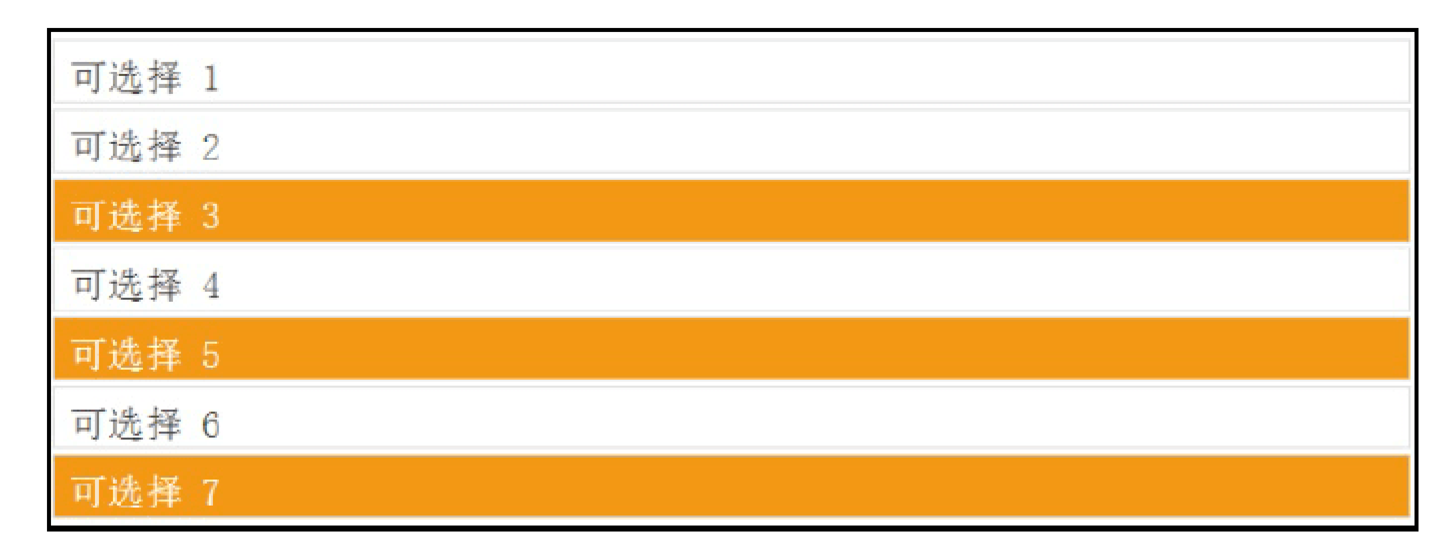


图 8-14　jQuery UI 可选择

4. sortable

sortable 行为使得 DOM 元素集合可以进行排序，当用户进行拖放排序时，其他元素的位置会自动调整。例如：

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery UI sortable - Default functionality</title>
  <script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
  <link rel="stylesheet" href="jQuery-ui-1.11.4.custom/jQuery-ui.min.css">
  <script src="jQuery-ui-1.11.4.custom/jQuery-ui.min.js"></script>
  <style>
  #sortable { list-style-type: none; margin: 0; padding: 0; width: 60%; }
  #sortable li { margin: 0 3px 3px 3px; padding: 0.4em; padding-left: 1.5em;
```

```
  font-size: 1.4em; height: 18px; }
 #sortable li span { position: absolute; margin-left: -1.3em; }
 </style>
 <script>
 $(function() {
   $( "#sortable" ).sortable();
   $( "#sortable" ).disableSelection();
 });
 </script>
</head>
<body>
 <ul id="sortable">
 <li class="ui-state-default"><span class="ui-icon"></span>Item 1</li>
 <li class="ui-state-default"><span class="ui-icon"></span>Item 2</li>
 <li class="ui-state-default"><span class="ui-icon"></span>Item 3</li>
 <li class="ui-state-default"><span class="ui-icon"></span>Item 4</li>
 <li class="ui-state-default"><span class="ui-icon"></span>Item 5</li>
 <li class="ui-state-default"><span class="ui-icon"></span>Item 6</li>
 <li class="ui-state-default"><span class="ui-icon"></span>Item 7</li>
 </ul>
</body>
</html>
```

运行效果如图 8-15 所示。

Item 1
Item 3
Item 4
Item 2
Item 5
Item 6
Item 7

图 8-15　jQuery UI 排序

jQuery UI 包含了许多维持状态的小部件(Widget)，因此它与典型的 jQuery 插件使用模式略有不同。所有的 jQuery UI 小部件使用相同的模式，所以只要我们学会使用其中一个，就知道如何使用其他的小部件。

5. Dialog 部件

对话框(Dialog)部件对于常见的 JavaScript 警告框而言，它由一个标题栏和一个内容区域组成，且可以移动位置和调整尺寸，默认可通过图标关闭。这种更灵活、可自定义主题的弹出框带来了更好的用户体验。例如：

```
<head>
<script>
 $(function() {
   $( "#dialog" ).dialog();
```

```
  });
  </script>
</head>
<body>
  <div id="dialog" title="基本的对话框">
  <p>这是一个默认的对话框，用于显示信息。对话框窗口可以移动，调整尺寸，默认可通过 'x' 图
     标关闭。</p>
</div>
 </body>
```

运行效果如图 8-16 所示。

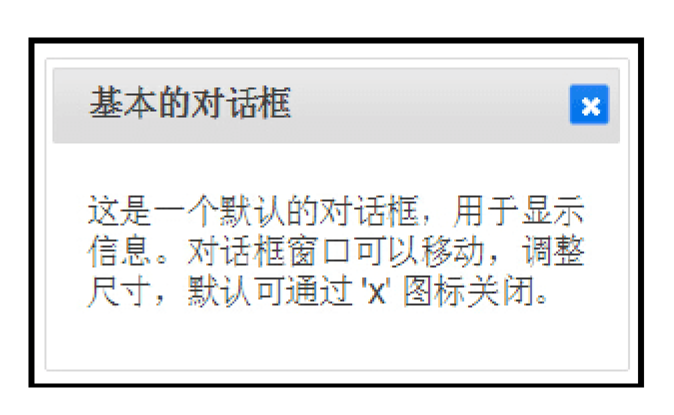


图 8-16　jQuery UI 对话框

6. Tooltip 部件

工具提示框(Tooltip)部件是可自定义的、可主题化的，并可以替代原生的工具提示框。该部件接受 tigle 和 alt 属性。例如：

```
 <head>
  <script>
   $(function() {
    $( document ).tooltip();
  });
  </script>
</head>
<body>
<p><label for="age">您的年龄：</label><input id="age"
   title="年龄仅用于统计。"></p>
 </body>
```

运行效果如图 8-17 所示。



图 8-17　jQuery UI 提示框

7. Datapicker 部件

日期选择(Datapicker)部件是一个富有交互性的部件，很可能是提供功能最多而代码量最少的部件。日期选择器绑定到一个标准的表单 input 字段上。把焦点移到 input 上(点击或者使用 tab 键)，会在一个小的覆盖层上打开一个交互日历。选择一个日期，点击页面上的任意地

方(输入框即失去焦点)，或者点击 Esc 键来关闭。如果选择了一个日期，则在 input 输入框中显示选择的值。例如：

```
<head>
<script>
   $(function() {
    $( "#datepicker" ).datepicker();
  });
  </script>
</head>
<body>
 <p>日期：<input type="text" id="datepicker"></p>
</body>
```

运行效果如图 8-18 所示。

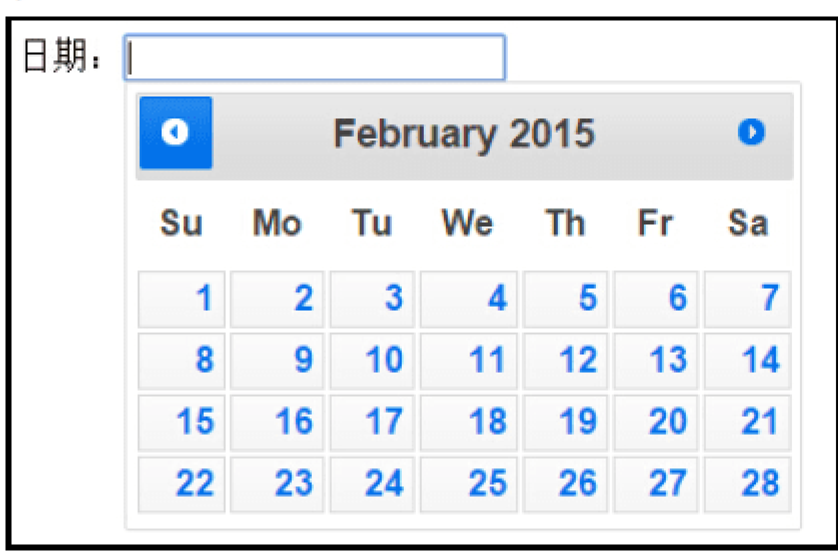


图 8-18　jQuery UI 日期

8. Autocomplete 部件

自动完成(Autocomplete)部件使得用户在完成表单时，能够看到一个可供选择的提示内容显示。数据源是一个简单的 JavaScript 数组，也可以使用 source 选项指定服务器提供的实时数据。例如：

```
<head>
$( "#tags" ).autocomplete({
      source: availableTags
    });
  });
  </script>
</head>
<body>
  <label for="tags">标签：</label>
  <input id="tags">
</body>
```

运行效果如图 8-19 所示。

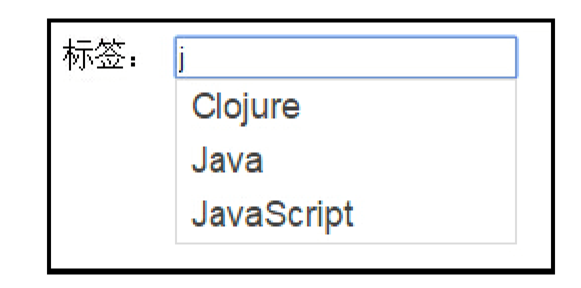


图 8-19　jQuery UI 自动完成

9. Tabs 部件

标签页(Tabs)部件可以创建一系列的选项卡式元素，下面的例子在目标 Div 元素中包含了数个<a>元素的<ul>元素，每个<a>元素带有一个指向页面内容区域的 href 属性。例如：

```
<body>
  <script>
  $(function(){
   $( "#tabs" ).tabs();
   });
  </script>
</head>
<body>

<div id="tabs">
  <ul>
    <li><a href="#fragment-1"><span>一</span></a></li>
    <li><a href="#fragment-2"><span>二</span></a></li>
    <li><a href="#fragment-3"><span>三</span></a></li>
  </ul>
  <div id="fragment-1">
    <p>第一个标签是默认激活的：</p>
    <pre><code>$( "#tabs" ).tabs(); </code></pre>
  </div>
  <div id="fragment-2">
     内容 22222222
  </div>
  <div id="fragment-3">
     内容 33333333
  </div>
</div>
```

运行效果如图 8-20 所示。



图 8-20　jQuery UI 标签页

10. Menu 部件

菜单(Menu)部件用于创建一个可折叠的菜单，它能够把一系列嵌套<ul>元素的结构化标记转化为一个可设置主体的菜单。例如：

```
<head>
  <script>
    $(function() {
```

```
      $( "#menu" ).menu();
    });
  </script>
  <style>
  .ui-menu { width: 150px; }
  </style>
</head>
<body>
<ul id="menu">
  <li><a href="#"><span class="ui-icon ui-icon-disk"></span>保存</a></li>
  <li><a href="#"><span class="ui-icon ui-icon-zoomin"></span>放大</a></li>
  <li><a href="#"><span class="ui-icon ui-icon-zoomout"></span>缩小</a></li>
  <li class="ui-state-disabled"><a href="#"><span class="ui-icon ui-icon-
    print"></span>打印...</a></li>
  <li>
    <a href="#">播放</a>
    <ul>
      <li><a href="#"><span class="ui-icon ui-icon-seek-start"></span>上一个
        </a></li>
      <li><a href="#"><span class="ui-icon ui-icon-stop"></span>停止</a></li>
      <li><a href="#"><span class="ui-icon ui-icon-play"></span>播放</a></li>
      <li><a href="#"><span class="ui-icon ui-icon-seek-end"></span>下一个
        </a></li>
    </ul>
  </li>
</ul>
</body>
```

运行结果如图 8-21 所示。

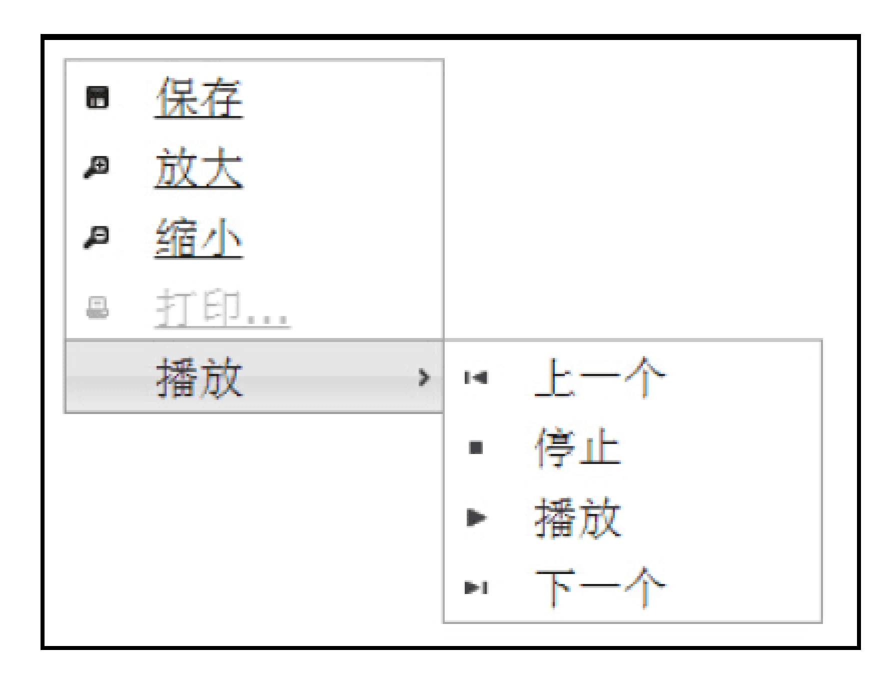


图 8-21　jQuery UI 菜单

11. Button 部件

按钮(Button)部件能够将<input>元素、<button>元素以及链接转化为浏览器兼容的按钮。例如：

```
<input type="submit" value="一个提交按钮">
$( "input[type=submit]").button();
```

12. Progressbar 部件

进度条(Progressbar)部件类似于<progress>元素，比如，我们在 id 为 elem 的元素上调用无参数的 progressbar() 方法，小部件会按照它的默认选项进行初始化。我们可以在初始化时传递一组选项(option 对象)，这样即可重写默认选项。代码如下：

```
$( "#elem" ).progressbar({ value: 20 });
```

既然小部件已经初始化，我们就可以查询它的状态，或者在小部件上执行动作。所有初始化后的动作都以方法调用的形式进行。为了在小部件上调用一个方法，我们可以向 jQuery 插件传递方法的名称。例如，为了在进度条小部件上调用 value 方法，应该使用下列代码：

```
$( "#elem" ).progressbar( "value" );
$( "#elem" ).progressbar("option", "value" );
```

13. Slider 部件

滑块(Slider)部件与 HTML5 中 type 类型为 range 的输入控件类似，我们可以访问滑动条的值。代码如下：

```
<script>
    $(document).ready(function() {
      $('#sld_value').slider({ min: 1, max: 100 });
      $("#dlg_value").dialog( { autoOpen: false, title: "Slider Value",
       open: function(){
         $("#dlg_value").html("The current value is: " +
             $("#sld_value").slider( "option", "value" ));
       }
     } );
    $("#btn_value").button().click(function()
    { $("#dlg_value").dialog("open"); });
    });
  </script>
</head>
<body>
       <div id="sld_value" style="margin-bottom:10px;"></div>
       <button id="btn_value">Display Value</button>
       <div id="dlg_value"></div>
</body>
```

运行效果如图 8-22 所示。

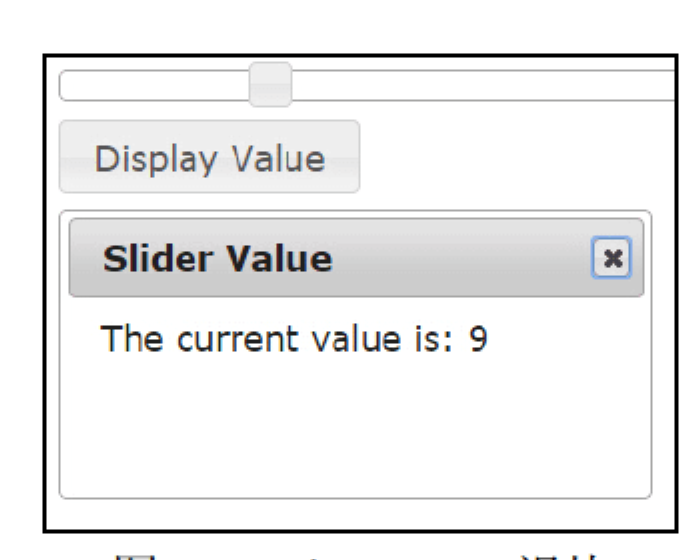


图 8-22　jQuery UI 滑块

本章小结

本章介绍了 jQuery 插件的使用方法，着重讨论了 jQuery ImageArea Select 插件和 jQuery templating 插件。

探讨了在选择第三方插件时的注意事项和原则，这也有助于自己开发一个流行的插件。还学习了创建插件的各种方式，在这个过程中既使用了 jQuery 的全局函数，也使用了 jQuery 的原型对象。

通过 jQuery UI 插件的学习，我们体验了 jQuery UI 插件的强大功能。无论从外观还是功能上看，jQuery UI 提供的“开箱即用”的部件都非常类似于我们熟悉的桌面应用程序中的相应元素。使用 jQuery UI 能够创建先进的网站和 Web 应用。

本章练习

1. 如何创建自定义的 jQuery 插件？
2. jQuery UI 的作用是什么？
3. js 模板引擎是什么？

第9章　使用Ajax和jQuery Mobile

Ajax 的应用非常广泛，在 Web 应用中经常使用 Ajax 提交数据、加载数据。甚至在类似 jQuery Mobile 的 UI 框架中，其底层也使用了 Ajax 技术。

本章内容：

- 练习使用以 Ajax 方式加载数据、提交数据
- 如何使用 jQuery Mobile 开发 Web 移动应用

9.1　以 Ajax 方式加载数据

9.1.1　加载 XML 数据

Ajax 这个名词包含异步的 XML，可见 XML 作为数据交换格式的通用性。在早期，XML 的推出使 Web 和电子商务的开发人员激动不已，因为很多语言都有支持 XML 解析和序列化的函数，一旦我们熟练掌握了 XML，就可发现它正是解决许多令人感到棘手的问题的有力工具。在人人微博网站的个人信息页面开发中，需要实现省市级联下拉框，页面效果如图 9-1 所示。



图 9-1　省市级联下拉框

可以把省市信息保存在数据库或者文件里面，这里使用 XML 保存省市信息，省份的数据保存在 Provinces.xml 中，格式如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<Provinces>
  <Province ID="1" ProvinceName="北京市">北京市</Province>
  <Province ID="2" ProvinceName="天津市">天津市</Province>
```

```
  <Province ID="3" ProvinceName="河北省">河北省</Province>
  ..............
</Provinces>
```

城市信息保存在 Cities.xml 文件中，格式如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<Cities>
  <City ID="1" CityName="北京市" PID="1" ZipCode="100000">北京市</City>
  <City ID="2" CityName="天津市" PID="2" ZipCode="100000">天津市</City>
  <City ID="3" CityName="石家庄市" PID="3" ZipCode="050000">石家庄市</City>
  <City ID="4" CityName="唐山市" PID="3" ZipCode="063000">唐山市</City>
  <City ID="5" CityName="秦皇岛市" PID="3" ZipCode="066000">秦皇岛市</City>
  ....................
</Cities>
```

使用 Ajax 技术实现用户选择左边的下拉框的某个选项时，右边下拉框的选项随之改变，还要实现用户将省市信息保存后，下次重新打开页面能够正确显示自己的省市信息。由于省市信息一般不会发生变化，而且经常使用，所以可以一次性将城市信息加载出来。如果更在意网络带宽和传输速度，也可以只加载已经选择的省份的城市信息。在这里我们使用的方案是当需要加载城市信息的时候，一次性使用 Ajax 方式将所有城市信息加载到页面前端。

当页面初次加载时，如果从服务器端得到的数据已经包含省份信息，则右边的下拉框也要随之变动。我们使用 jQuery 事件的自定义事件，定义一个 provinceChange 事件，在城市下拉框上监听这个事件，然后显示不同的数据。JavaScript 代码如下：

```
//------------------加载省份 城市数据------------------------------------
   $("select[name=province]").append("<option value='0'>请选择..</option>");
   $("select[name=city]").append("<option value='0'>请选择..</option>");
   //省份，城市
   var provinceData,cityData;
   $.ajax({
    url: '/rrwb/xml/Provinces.xml',
    type: 'GET',
    dataType: 'xml'
   })
   .done(function(xmldoc) {
     console.log("load provinces success");
     provinceData = $(xmldoc).find("Province");
     $(provinceData).each(function(){
      $("select[name=province]").append("<option value='"+$(this).attr("ID")
         +"'>"+$(this).attr("ProvinceName")+"</option>");
     })
   })

   $("select[name=province]").change(function(event) {
      $("select[name=city]").trigger( "provinceChange",[$(this).val(),0]);
    });
  $("select[name=city]").on("provinceChange",function(event,provinceValue,
     cityValue){
```

```
    //如果没有城市数据，则同步加载
    if(cityData == null){
      $.ajax({
        url:"/rrwb/xml/Cities.xml",
        type:"get",
        datatype:"xml",
        //同步执行，避免绑定失败
        async : false,
        success:function(xml){
          cityData = xml;
        }
       });
    }
    $("select[name=city]").find("option").remove();
    $("select[name=city]").append("<option value='0'>请选择..</option>");

   $(cityData).find("Cities>City[PID="+provinceValue+"]").each(function(){

    $("select[name=city]").append("<option value='"+$(this).attr("ID")+
      "'>"+$(this).attr("CityName")+"</option>");
         });
         $("select[name=city]").val(cityValue);

  });
```

总之，XML 提供了一种开发 Web 应用程序具有潜力和灵活性的技术，它使应用程序可以实现异构环境下的无缝集成。

9.1.2　加载 JSON 数据

在人人微博网站中，服务器从数据库返回的数据都是以 JSON 数据格式返回的。当用户访问个人信息页面时，首先会加载用户的个人信息。代码如下所示：

```
//--------------------加载用户数据---------------------------------------------
 $.ajax({
    url: 'http://localhost:8080/rrwb/crud/user/show',
    type: 'POST',
    dataType: 'json',
    data: {_id: sessionStorage.userId},
  })
  .done(function(response) {
          $('input[name=_id]').val(response._id);
          $('input[name=login_name]').val(response.login_name);
          $('input[name=nick_name]').val(response.nick_name);
          $('input[name=real_name]').val(response.real_name);
          $('select[name=province]').val(response.provinceValue);
          //触发 provinceChange 事件
          //$("select[name=city]").trigger( "provinceChange",[response.
               provinceValue,response.cityValue]);
          $('input[name=birthday]').val(response.birthday);
```

```
            $('input[name=blog]').val(response.blog);
            $('input[name=qq]').val(response.qq);
            $('input[name=msn]').val(response.msn);
           $('input[name=gender][value='+response.gender+']').attr("checked",
            'checked');
            $("#crop_preview").removeAttr("style");
            $("#crop_preview").attr("src", response.img) ;
            $("#age").val(response.age);
            $("#display_age").val(response.age);
            $("#other_email").val(response.other_email);
    })
  //-------------------------------------------------------------------------
```

9.2　以 Ajax 方式提交数据

9.2.1　无刷新的表单修改

当用户修改个人信息后，需要将个人信息保存到服务器。一方面我们需要知道服务器保存个人信息的 API，如下所示：

接口说明：编辑用户个人信息

URL：http://localhost:8080/rrwb/crud/user/edit

HTTP 请求方式：POST/GET

请求参数如表 9-1 所示。

表 9-1　请求参数

参 数 名 称	类　型	说　明
login_name	String	登录名，必输
nick_name	String	昵称，必输
img	String	缩略图地址
real_name	String	真实姓名
provinceValue	String	所在省份编号
provinceName	String	所在省份名称
cityValue	String	所在城市编号
cityName	String	所在城市名称
gender	String	性别
age	String	年龄
other_email	String	备用邮箱
blog	String	博客地址
qq	String	QQ
pwd	String	密码，必输
msn	String	MSN
birthday	String	生日

响应参数如表 9-2 所示。

表 9-2　响应参数

参数名称	类　型	描　述
code	Number	0：插入失败 1：插入成功

我们再次确定页面中个人信息的相关字段与 API 接口一致，然后使用 Ajax 进行表单提交。实现此功能的代码如下所示：

```
//-------------------保存提交-------------------------------------------
$("#send").click(function(e) {
     e.preventDefault();
     //省份名称、城市名称
     var myPName = '',myCName = '';
     //检查下拉框选择的不是“请选择”
     if($('select[name=province]').val() != '0'){
         myPName = $('select[name=province] option:selected').text();
     }
     if($('select[name=city]').val() != '0'){
         myCName = $('select[name=city] option:selected').text();
     }
     $.ajax({
       url: 'http://localhost:8080/rrwb/crud/user/edit',
       type: 'post',
       dataType: 'json',
       data: {_id:$('input[name=_id]').val() ,
            login_name:$('input[name=login_name]').val(),
            nick_name:$('input[name=nick_name]').val(),
            real_name:$('input[name=real_name]').val(),
            provinceValue:$('select[name=province]').val(),
            provinceName:myPName,
            cityValue:$('select[name=city]').val(),
            cityName:myCName,
            birthday:$('input[name=birthday]').val(),
            img:$("#crop_preview").attr("src"),
            msn:$('input[name=msn]').val(),
            qq:$('input[name=qq]').val(),
            blog:$('input[name=blog]').val(),
            other_email:$('input[name=other_email]').val(),
            gender:$('input[type=radio]:checked').val(),
            age:$('input[name=age]').val()}
     })
     .done(function(response) {
       if(response.code == 1 ){
         alert("保存成功！");
         console.log(response.data);
       }
       console.log("success");
```

```
        })
    });
```

9.2.2　无刷新的表单新增

在个人主页页面中，需要实现无刷新的微博发布，也就是无刷新的表单新增。新增成功后，在微博列表中显示所有最新发布的微博，如图 9-2 所示。

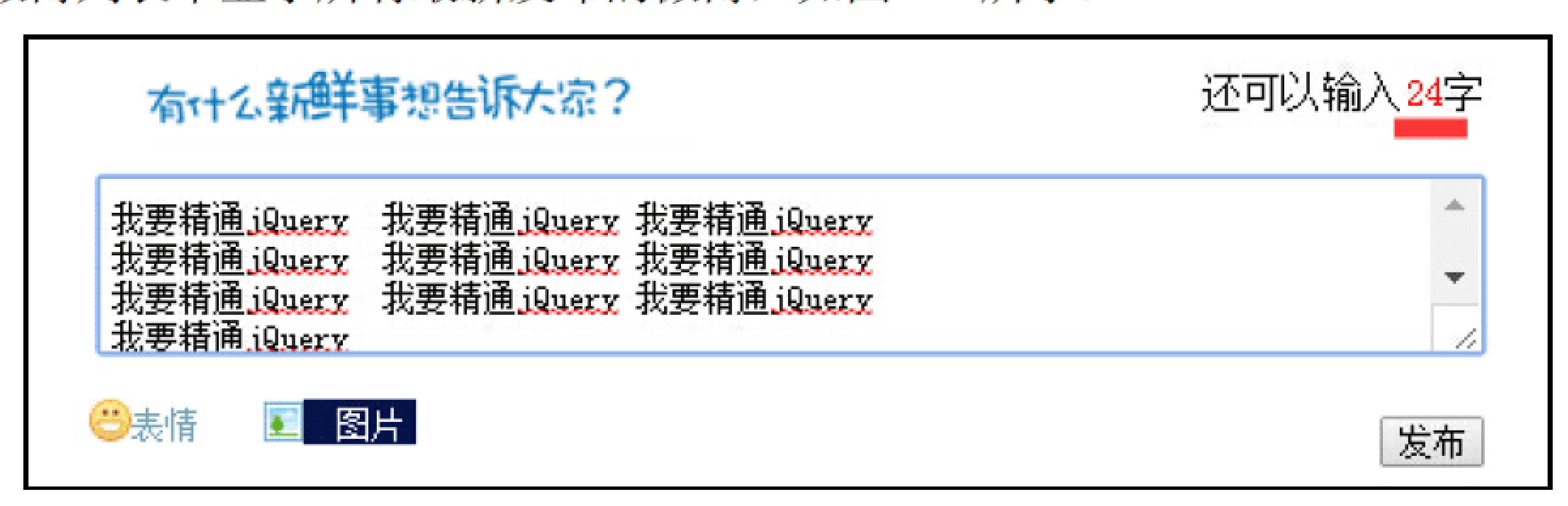


图 9-2　发布微博

这里需要调用服务端新增微博的 API，如下所示：

接口说明：发布微博

URL：http://localhost:8080/rrwb/crud/weibo/add

HTTP 请求方式：POST/GET

请求参数如表 9-3 所示。

表 9-3　请求参数

参 数 名 称	类　　型	说　　明
user_id	Number	用户 id，必输
user_nickName	String	必输
user_head_img	String	
img	String	
content	String	必输

响应参数如表 9-4 所示。

表 9-4　响应参数

参 数 名 称	类　　型	说　　明
code	Number	0：插入失败 1：插入成功

下面我们使用 Ajax 来实现无刷新的表单新增。

HTML5 代码如下：

```
<div id="tips">
  <div id="look"><img src="images/ico_01.jpg" /><span class="look1">表情
    </span></div>
```

```
<form name="" id="upload_form" method="post" enctype="multipart/form-data" >
<div id="picture"><img src="images/ico_02.jpg" />
<span class="look1">
<!--上传图片按钮-->
<div class="fileWrapper">
<input name="photo" id="file" class="fileInput" size="27" type="file" />
<a class="fileButton">图片</a>
</div>
</span>
</div>
<!--提示信息-->
<div id="notice" ></div>
<!--保存图片地址-->
<input type="hidden" id="uploaded_image"  name="uploaded_image" />
</form>
<div id="submit">
<div id="sub"><input id = 'submitWeiboBtn' name="submit" type="button"
  value="发布" />
</div>
</div>
</div>
```

JavaScript 代码如下：

```
// 默认配置
var defaults = {
  allowed: 140,
  warning: 25,
  cssWarning: 'warning',
  //微博的总页数
  weiboPages:0,
  //当前显示第几页微博
  currentWeiboPage:1,
  //每一页的微博个数
  currentWeiboSize:6,
  friendPages:0,
  //上次获取最新微博的时间
  last_refresh_time:(new Date()).getTime(),
  //微博列表中时间最晚的微博的发布时间
  last_more_time:(new Date()).getTime()
};
//-----------上传微博图片---------------------------------------
  $("#file").change(function(event) {
     if($("#file").val()!=''){
     $('#notice').text('正在上传').fadeIn();
     $("#upload_form").submit();
    }
   else {
     $('#notice').hide();
    }
  });
```

```
$("#upload_form").submit(function(e) {
     e.preventDefault();
    if(window.FormData !== undefined)  // HTML5 浏览器
    {
        var formData = new FormData(this);
       //上传图片
        $.ajax({
            url: 'http://localhost:8080/rrwb/crud/uploadWeiboServlet',
            type: 'POST',
            mimeType:"multipart/form-data",
            data:formData,
            contentType: false,
        cache: false,
        processData:false
        })
        .done(function(data) {
            console.log(data);
            var response = $.parseJSON(data);
            if(response.code == 1){
              $('#notice').text(response.url)
              $("#uploaded_image").val(response.url);
              $('#notice').fadeOut();
              alert("上传成功！");
            }
        })
        .fail(function() {
            console.log("error");
        })
        .always(function() {
            console.log("complete");
        });
         //阻止默认动作
    }
   else  //旧浏览器
    {
        alert('浏览器不支持 HTML5');
    }
 });

//--------------------发布微博----------------------------------------------
$("#submitWeiboBtn").click(function(e) {
      e.preventDefault();
      $.ajax({
        url: 'http://localhost:8080/rrwb/crud/weibo/add',
        type: 'post',
        dataType: 'json',
        data: {user_id:sessionStorage.userId,
               user_nickName:$('#nick_name').text(),
              content:$('#weiboTextArea').val(),
              img:$("#uploaded_image").val(),
```

```
                user_head_img:$('.cont_ontleft img').attr('src')
              }
          })
          .done(function(response) {
            if(response.code == 1 ){
              alert("保存成功！");
              $('#weiboTextArea').val('');
              $("#uploaded_image").val('');
              //将数字变为初值 140
              $(".aviableCount span").html('140');
            }
            console.log("success");
          })
          .fail(function() {
            console.log("发布微博 error");
          })

      });
```

9.2.3　无刷新的列表数据

当个人主页发布微博成功后，需要加载最新发布的微博并显示在微博列表的最前面。这里需要调用服务端查询最新发布的微博的 API，如下所示：

- 接口说明：查询尚未加载的最新发布的微博
- URL：http://localhost:8080/rrwb/crud/weibo/ latest_weibos
- HTTP 请求方式：POST/GET

请求参数如表 9-5 所示。

表 9-5　请求参数

参 数 名 称	类　型	说　明
user_id	Number	用户 id，必输
page	String	当前页数，从 1 开始
size	String	每页显示的数量
currentTime	Number	上次加载最新微博的时间

响应参数如表 9-6 所示。

表 9-6　响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	Array	查询到的数据的数组
rows	String	查询到的记录总数量

下面我们使用 Ajax 无刷新的获取列表数据，代码如下：

```
//加载所有自己及关注者的比当前时间新的微博
  function load_Latest_Followed_Weibos(){
    $.ajax({
      url: 'http://localhost:8080/rrwb/crud/weibo/latest_weibos',
      type: 'post',
      dataType: 'json',
      data: {user_id: sessionStorage.userId,page:1,size:100,currentTime:
        defaults.last_refresh_time},
    })
    .done(function(response) {
      if(response.code == 1 && response.rows > 0){
        var weibos = response.data;
        defaults.weiboPages=response.pages;
        $("#weibos").loadTemplate("#template", weibos,{prepend:true});
        defaults.last_refresh_time = (new Date()).getTime();
      }
      console.log("success");
    })
    .fail(function() {
      console.log("error");
    })
    .always(function() {
      console.log("complete");
    });
  }
```

9.2.4　加载更多列表数据

在微博列表中，可以通过单击“更多”按钮加载以前发布的微博，如图 8-6 所示。这里需要调用服务端查询较早发布的微博的 API，如下所示：

- 接口说明：查询较早发布的微博
- URL：http://localhost:8080/rrwb/crud/weibo/previous_weibos
- HTTP 请求方式：POST/GET

请求参数如表 9-7 所示。

表 9-7　请求参数

参 数 名 称	类　型	说　明
user_id	Number	用户 id，必输
page	String	当前页数，从 1 开始
size	String	每页显示的数量
last_More_Time	Number	指定从哪个时间节点向后查询微博

响应参数如表 9-8 所示。

表 9-8　响应参数

参数名称	类　型	说　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	Array	查询到的数据的数组
rows	String	查询到的记录总数量

下面我们使用 Ajax 实现加载更多列表数据的功能。代码如下：

```
//单击“更多”按钮，加载所有自己及关注者的较旧的微博
  function load_Previous_Weibos(){
   $.ajax({
     url: 'http://localhost:8080/rrwb/crud/weibo/previous_weibos',
     type: 'post',
     dataType: 'json',
     data: {
             user_id: sessionStorage.userId,
             page:defaults.currentWeiboPage,
             size:defaults.currentWeiboSize,
             last_More_Time:defaults.last_more_time
           }
    })
    .done(function(response) {
      if(response.code == 1 && response.rows > 0){
        var weibos = response.data;
        //添加到列表后面
        $("#weibos").loadTemplate("#template", weibos,{append:true});
        //如果行数小于 defaults.currentWeiboSize，则全部加载完后“更多”按钮消失
        if(weibos.length < defaults.currentWeiboSize){
          $('#moreList').fadeOut();
        }
        defaults.last_more_time = weibos[weibos.length-1].timestamp;
      }
      console.log("success");
    })
    .fail(function() {
      console.log("load_Previous_Weibos error");
    })
```

网站中可能有许多图片，这时页面中图片数量多，而且比较大。如果页面载入时一次性加载完毕，需要用户等半天，针对这种情况，目前很流行的做法就是滚动动态加载或者延迟加载。这种做法的好处，一是页面加载速度快，二是节约了流量，因为不可能每个用户浏览页面时都从头滑动到尾。下面我们在微博列表中实现此功能。JavaScript 代码如下：

```
//-----------滚动加载图片--------------------------
    $.fn.isOnScreen = function(){
    var win = $(window);
    //视窗的边框位置
```

```
    var viewport = {
      top : win.scrollTop(),
      left : win.scrollLeft()
    };
    viewport.right = viewport.left + win.width();
    viewport.bottom = viewport.top + win.height();
    //当前元素的边框位置
    var bounds = this.offset();
    bounds.right = bounds.left + this.outerWidth();
    bounds.bottom = bounds.top + this.outerHeight();
    //判断元素是否在视窗内
    return (!(viewport.right < bounds.left || viewport.left > bounds.right ||
       viewport.bottom < bounds.top || viewport.top > bounds.bottom));
  }

  function lazyload() {
    $('.lazyload').each(function() {
      var element = $(this);
      if (element.isOnScreen()) {
        //可以使用 data-* 属性，而不使用 value
        element.attr('src', element.attr('value'));
        element.removeClass('lazyload');
      }
    })
  }
  lazyload();
  $(window).scroll(function() {
    lazyload();
  })
```

在 HTML 页面中只要在 img 元素上使用 lazyload 类，就可以了。例如：

```
<img class="lazyload zoom" data-value="img" onclick="zoom_image($(this).
   parent());"/>
```

9.3　jQuery Mobile 介绍

9.3.1　jQuery Mobile 特性

jQuery Mobile 以“Write Less，Do More”作为目标，为所有的主流移动操作系统平台提供了高度统一的 UI 框架：jQuery 的移动框架可以让我们为所有流行的移动平台设计一个高度定制和品牌化的 Web 应用程序，而不必为每个移动设备编写单独的应用程序。

jQuery Mobile 目前支持的移动平台有苹果公司的 iOS(iPhone、ipad、iPod Touch)、Android、BlackBerry OS 6.0、Windows Mobile、Symbian 和 MeeGo 等众多移动平台。

目前 jQuery Mobile 的特性包括：

- 与jQuery桌面版一致的jQuery核心和语法，以及最小的学习曲线。
- 兼容所有主流的移动平台以及所有支持HTML的移动平台。
- 标记驱动的配置 jQuery Mobile 采用完全的标记驱动而不需要JavaScript的配置，只要了解HTML5就可以创建丰富的Web应用。
- jQuery Mobile采用完全的渐进增强原则：通过一个全功能的HTML网页和额外的JavaScript功能层，提供顶级的在线体验。这意味着即使移动浏览器不支持JavaScript，我们的jQuery Mobile应用程序中不强制使用JavaScript，移动应用程序也仍能正常使用。
- 支持Ajax动态加载内容。
- 简单的API为用户提供鼠标、触摸和光标焦点简单的输入法支持。
- 强大的主题化框架jQuery Mobile提供强大的主题化框架和UI接口。

jQuery Mobile 这个组合容易引起一些混淆，这里给出 jQuery Mobile 不是什么。

- 不是在移动浏览器上的jQuery的替代品。
- 不是打包为原生Web应用(Native Web App)的工具。
- 不是为JavaScript爱好者提供的框架。

总之，jQuery Mobile 只是一个 UI 框架，官网地址为 http://jQuerymobile.com/，如图 9-3 所示。

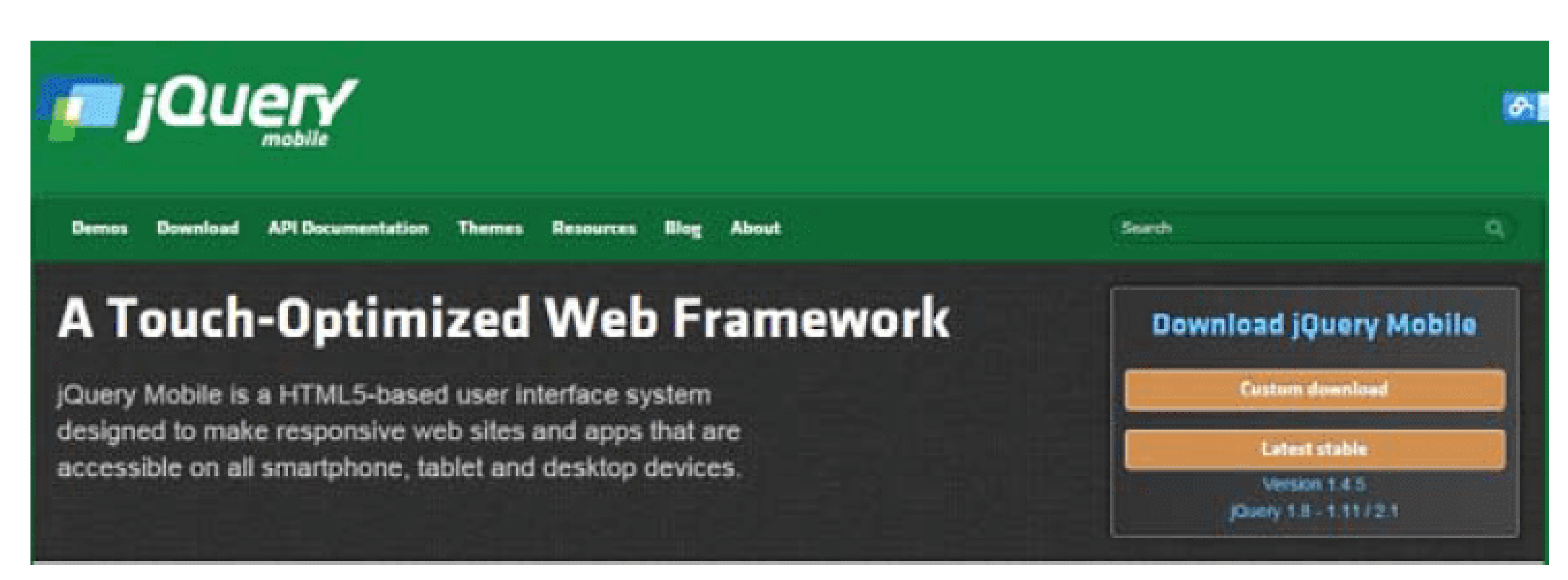


图 9-3　jQuery Mobile 官网

下载完成后，解压后如图 9-4 所示。

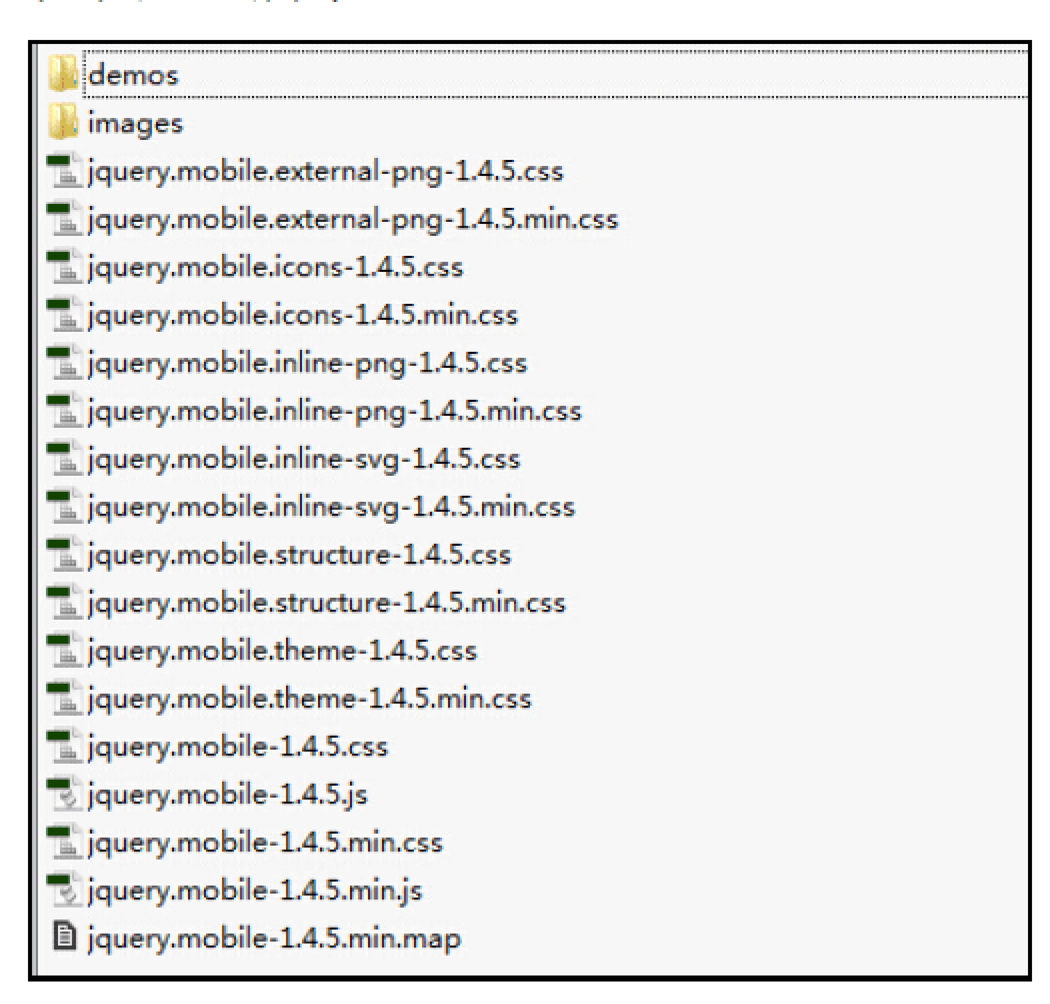


图 9-4　jQuery Mobile 文件

9.3.2　jQuery Mobile 初始配置

只需直接在 HTML 页面中引用以下样式表和 JavaScript 库，这样 jQuery Mobile 就可以工作了：

```
<head>
<link rel=stylesheet href="jQuery.mobile-1.4.5.css">
<script src="jQuery.js"></script>
<script src="jQuery.mobile-1.4.5.js"></script>
</head>
```

或者从 CDN 引用 jQuery Mobile

```
<link rel="stylesheet" href="http://code.jQuery.com/mobile/latest/jQuery.
  mobile.min.css" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<script src="http://code.jQuery.com/mobile/latest/jQuery.mobile.min.js">
  </script>
```

9.4　jQuery Mobile 页面和对话框

9.4.1　页面基础

页面是 jQuery Mobile 的主要容器和入口，典型的页面包括三部分：header、content 和 footer。例如：

```
<body>
<div data-role="page">
  <div data-role="header">
    <h1>欢迎访问我的主页</h1>
  </div>
  <div data-role="content">
    <p>我是一名移动开发者！</p>
  </div>
  <div data-role="footer">
    <h1>页脚</h1>
  </div>
</div>
</body>
```

运行效果如图 9-5 所示。

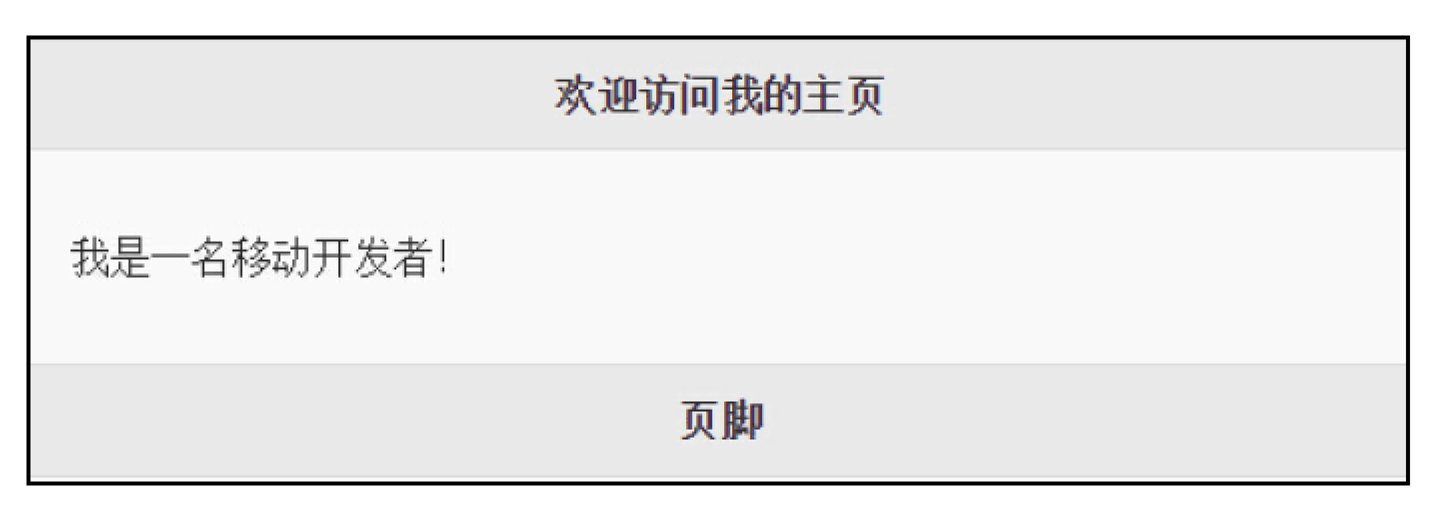


图 9-5　jQuery Mobile 页面基础

jQuery Mobile 中使用了 div 元素，而没有使用 HTML5 中的 header、footer 元素，这是为了兼容不支持 HTML5 的浏览器。如果我们确定用户一定使用 HTML5 浏览器，也可以使用 header、footer 等元素。

jQuery Mobile 在 div 中定义了 role。不同的 role 是通过 data-role 属性定义的。比如<div data-role="page">，以 data-开头的属性是 HTML5 提供的自定义属性。通过 data-role 属性的约定，jQuery Mobile 可以为我们把 div 元素渲染成更丰富、交互性更强的元素。

在上面的代码中，data-role="page" 是显示在浏览器中的页面；data-role="header" 创建页面上方的工具栏(常用于标题和搜索按钮)；data-role="content" 定义页面的内容，比如文本、图像、表单和按钮等；data-role="footer" 创建页面底部的工具栏。

为了适应手机的屏幕大小，在 2010 年，Ethan Marcotte 提出了“自适应网页设计”(Responsive Web Design)这个名词，指可以自动识别屏幕宽度并做出相应调整的网页设计。通常手机的屏幕比较小，宽度通常在 600 像素以下；PC 的屏幕宽度一般都在 1000 像素以上(目前主流宽度是 1366×768)。比如在图 9-6 所示的自适应设计中。



图 9-6　自适应设计

如果屏幕宽度大于 1300 像素，则 6 张图片并排在一行。如果屏幕宽度在 600 像素到 1300 像素之间，则 6 张图片分成两行。如果屏幕宽度在 400 像素到 600 像素之间，则导航栏移到网页头部。这就需要在网页代码的头部，加入一行 viewport 元标签。

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

viewport 是网页默认的宽度和高度，上面这行代码的意思是，网页宽度默认等于屏幕宽度(width=device-width)，原始缩放比例为 1.0(initial-scale=1)，即网页初始大小占屏幕面积的 100%。

9.4.2　创建多个页面

在 jQuery Mobile 中，我们可以在单一 HTML 文件中创建多个页面。请通过唯一的 id 来分隔每张页面，并使用 href 属性来连接彼此，例如：

```
 <div data-role="page" id="pageone">
  <div data-role="header">
    <h1>欢迎访问我的主页</h1>
  </div>
  <div data-role="content">
    <p>Welcome!</p>
    <a href="#pagetwo">转到页面二</a>
  </div>

  <div data-role="footer">
  <h1>页脚</h1>
  </div>
</div>
<div data-role="page" id="pagetwo">
  <div data-role="header">
    <h1>欢迎访问我的主页</h1>
  </div>
  <div data-role="content">
    <p>Goodbye!</p>
    <a href="#pageone">转到页面一</a>
  </div>
  <div data-role="footer">
  <h1>页脚</h1>
  </div>
</div>
```

运行效果如图 9-7 所示。



图 9-7　jQuery Mobile 的多个页面

9.4.3　将页面用作对话框

对话框是用来显示信息或请求输入的视窗类型。

如需在用户点击(轻触)链接时创建一个对话框，请向该链接添加 data-rel="dialog"，例如：

```
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>欢迎访问我的主页</h1>
  </div>
  <div data-role="content">
    <p>欢迎！</p>
    <a href="#pagetwo" data-rel="dialog">转到页面二</a>
  </div>

  <div data-role="footer">
  <h1>页脚文本</h1>
  </div>
</div>
<div data-role="page" id="pagetwo">
  <div data-role="header">
    <h1>我是一个对话框！</h1>
  </div>
  <div data-role="content">
<p>对话框与普通页面不同，它显示在当前页面的顶端。它不会横跨整个页面宽度。对话框页眉中的图
    标 “X” 可关闭对话框。</p>
    <a href="#pageone">转到页面一</a>
  </div>
  <div data-role="footer">
  <h1>页脚</h1>
  </div>
</div>
```

运行效果如图 9-8 所示。

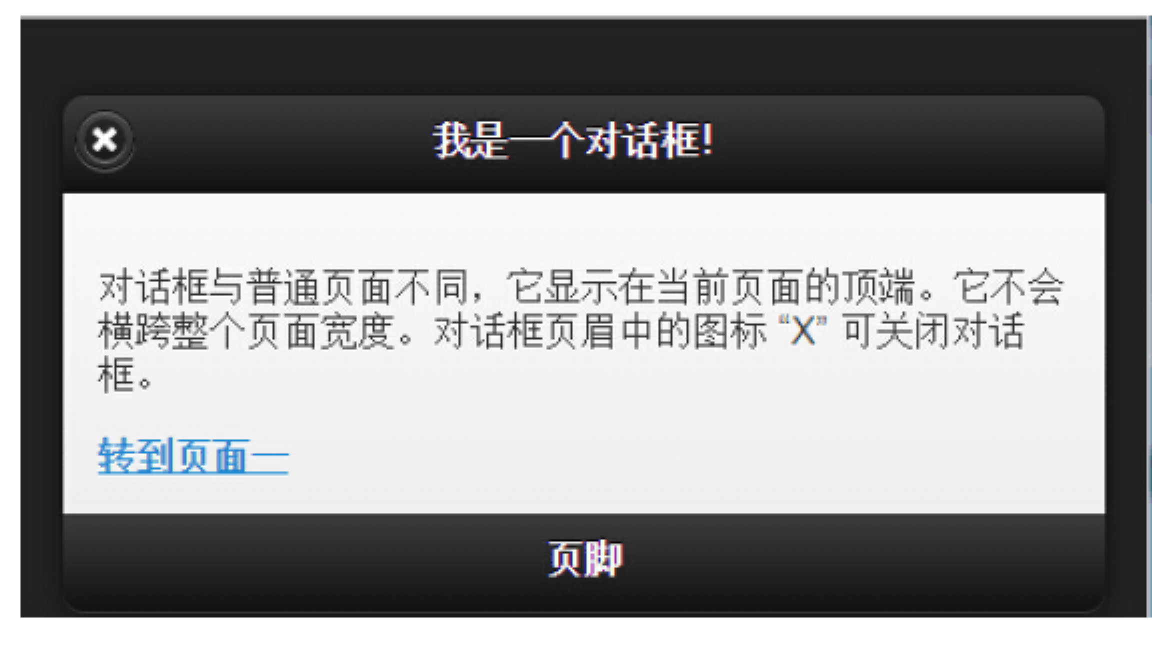


图 9-8　jQuery Mobile 对话框

9.5　jQuery Mobile 表单

9.5.1　jQuery Mobile 表单输入

jQuery Mobile 会自动为 HTML 表单添加优异的便于触控的外观。表单在移动设备上的显示效果如图 9-9 所示。



图 9-9　jQuery Mobile 表单

按钮在表单中经常使用，jQuery Mobile 提供了表单的组件以增强按钮的用户体验。我们可以在按钮上添加图片或者文字。创建按钮有如下几种方式:

- 使用 button 元素，使用 type 属性为 button、submit、reset、image 的输入元素
- 使用 a 元素，并添加 data-role="button"属性
- header 或 footer 中的 a 元素被自动渲染为按钮

例如:

```
<div data-role="page" id="pageone">
  <div data-role="header">
    <h1>欢迎访问我的主页</h1>
    <a href="#"  data-icon="refresh">Text with icon</a>
  </div>
  <div data-role="content">
    <a href="#" data-role="button" data-corners="false">No rounded corners</a>
    <a href="#" data-role="button" data-inline="true">Inline button</a>
    <a href="#" data-role="button" data-icon="refresh" data-iconpos="notext">
  Only icon</a>
    <input type="submit" name="submit" value="登录"  id="login_btn"/>
    <button>注册</button>
  </div>
</div>
```

HTML5 中的其他输入元素可以直接在 jQuery Mobile 中使用。jQuery Mobile 会为它们设置专门针对移动设备的美观易用的样式，比如文本框会自动扩大。在实现表单代码时，都会有一个 label 元素对应一个 input 元素，例如：

```
<div data-role="content">
   <div data-role="fieldcontain">
    <label for="fullname">name: </label>
   <input type="text" name="fullname" id="fullname">
  </div>
  <div data-role="fieldcontain">
     <label for="search">Search:</label>
     <input type="search" name="search" id="search">
  </div>
  <fieldset data-role="controlgroup">
   <legend>Choose your gender:</legend>
     <label for="male">Male</label>
     <input type="radio" name="gender" id="male" value="male">
     <label for="female">Female</label>
     <input type="radio" name="gender" id="female" value="female">
  </fieldset>
  <fieldset data-role="controlgroup" data-type="horizontal">
   <legend>Choose as many favorite colors as you'd like:</legend>
     <label for="red">Red</label>
     <input type="checkbox" name="favcolor" id="red" value="red">
     <label for="green">Green</label>
     <input type="checkbox" name="favcolor" id="green" value="green">
     <label for="blue">Blue</label>
     <input type="checkbox" name="favcolor" id="blue" value="blue">
  </fieldset>
  <fieldset data-role="controlgroup">
     <label for="slider-1">Slider:</label>
     <input type="range" name="slider-1" id="slider-1" min="0" max="100"
        value="50">
  </fieldset>
  </div>
```

上面的代码中，data-role="fieldcontain"是为了将内容进行分块，主要的功能就是在每个区域后面加上一条横线，便于区分区域。data-role="controlgroup"是为了使其所包围的内容更加紧凑，更像一个整体，使用 data-type="horizontal" 属性可以对组内的元素进行水平分组，也就是横向排列。运行效果如图 9-10 所示。

图 9-10　jQuery Mobile 表单内容分组

9.5.2　jQuery Mobile 表单选择

在 jQuery Mobile 中也可以使用选择菜单，使用<select>元素创建带有若干选项的下拉菜单，使用<option>元素定义列表中的可用选项。在 jQuery Mobile 中还可以通过滑动开关的方式进行选择。例如：

```
<fieldset data-role="fieldcontain">
    <label for="day">选择日期</label>
    <select name="day" id="day">
      <option value="mon">星期一</option>
      <option value="tue">星期二</option>
      <option value="wed">星期三</option>
    </select>
  </fieldset>
  <fieldset data-role="fieldcontain">
     <label for="yesNo">是否:</label>
      <select name="yesNo" id="yesNo" data-role="slider" >
        <option value="0">是</option>
        <option value="1">否</option>
      </select>
  </fieldset>
```

显示效果如图 9-11 所示。



图 9-11　jQuery Mobile 表单选择

9.6　jQuery Mobile 列表

9.6.1　jQuery Mobile 列表视图

jQuery Mobile 中的列表组件有许多新的特性，应用 data-role="listview"到 ul 或者 ol 元素上，就可以实现功能强大的无序列表和有序列表。例如：

```
<ul data-role="listview" data-filter="true" data-filter-placeholder="搜索国家" >
    <li data-role="list-divider">A 组</li>
    <li><a href="#">巴西</a></li>
    <li><a href="#">喀麦隆</a></li>
    <li><a href="#">墨西哥</a></li>
    <li><a href="#">克罗地亚</a></li>
    <li data-role="list-divider">B 组</li>
    <li><a href="#">西班牙</a></li>
    <li><a href="#">智利</a></li>
    <li><a href="#">澳大利亚</a></li>
    <li><a href="#">荷兰</a></li>
  </ul>
```

在上面的代码中，使用 data-filter="true"向列表中添加搜索框，输入关键字后列表会自动过滤符合关键字的数据项。还可以使用 data-filter-placeholder="搜索国家"设置搜索框中的默认文本，如果需要为列表添加圆角和外边距，可以使用 data-inset="true" 属性。data-role="list-divider"指定当前行为分隔符，运行效果如图 9-12 所示。



图 9-12　jQuery Mobile 列表

jQuery Mobile 列表还可以自动分组，添加 data-autodividers="true"属性后，列表会根据列表的字母顺序进行分组。例如：

```
<ul data-role="listview" data-autodividers="true">
  <li><a href="#">Adam</a></li>
  <li><a href="#">Angela</a></li>
  <li><a href="#">Bill</a></li>
```

```
  <li><a href="#">Calvin</a></li>
</ul>
```

运行效果如图 9-13 所示。

图 9-13　jQuery Mobile 自动分组

如果在列表中嵌套了其他的列表，jQuery Mobile 会自动处理嵌套，并增加点击列表的效果。例如：

```
<ul data-role="listview" data-inset="true" data-shadow="false">
     <li data-role="collapsible" data-iconpos="right" data-inset="false">
       <h2>Birds</h2>
       <ul data-role="listview" data-theme="b" >
         <li><a href="#">Condor</a></li>
         <li><a href="#">Eagle</a></li>
         <li><a href="#">Sparrow</a></li>
       </ul>
     </li>
     <li> <a href="#">Humans</a></li>
     <li data-role="collapsible" data-iconpos="right" data-inset="false">
       <h2>Fish</h2>
       <ul data-role="listview" data-theme="b">
         <li><a href="#">Salmon</a></li>
         <li><a href="#">Pollock</a></li>
         <li><a href="#">Trout</a></li>
       </ul>
     </li>
  </ul>
```

运行效果如图 9-14 所示。



图 9-14　jQuery Mobile 列表嵌套

在 jQuery Mobile 中可以使用列表制作导航栏(Navbars)，导航栏一般被放置在页面的 header 或 footer 部分。例如：

```
<div data-role="navbar" data-grid="c">
     <ul>
       <li><a href="#" class="ui-btn-active">One</a></li>
       <li><a href="#">Two</a></li>
       <li><a href="#">Three</a></li>
       <li><a href="#">Four</a></li>
     </ul>
 </div>
```

运行效果如图 9-15 所示。

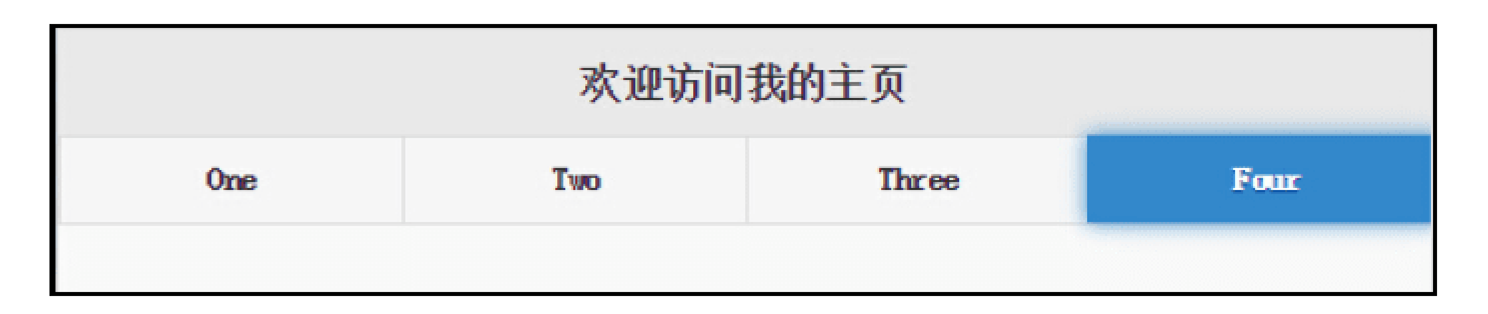


图 9-15　jQuery Mobile 导航栏

在上面的例子中，使用 data-role="navbar"增强 div 元素为导航栏，并使用 ui-btn-active 样式类设置当前激活的选项。

9.6.2　jQuery Mobile 列表内容

在每一个列表选项中，还可以添加更多的内容，例如添加缩略图和显示简单的统计数字等。这在移动应用中非常常见。例如：

```
<div data-role="page" id="pageone">
   <ul data-role="listview">
    <li>
     <a href="#"><img src="_assets/img/apple.png"></a>
     <a href="#">Some link</a>
```

```
    </li>
  </ul>
</div>
```

上面的代码在 li 元素中放置了两个链接，jQuery Mobile 会自动为第二个链接添加蓝色箭头图标的样式，链接中的文本将在用户划过该图标时显示，效果如图 9-16 所示。

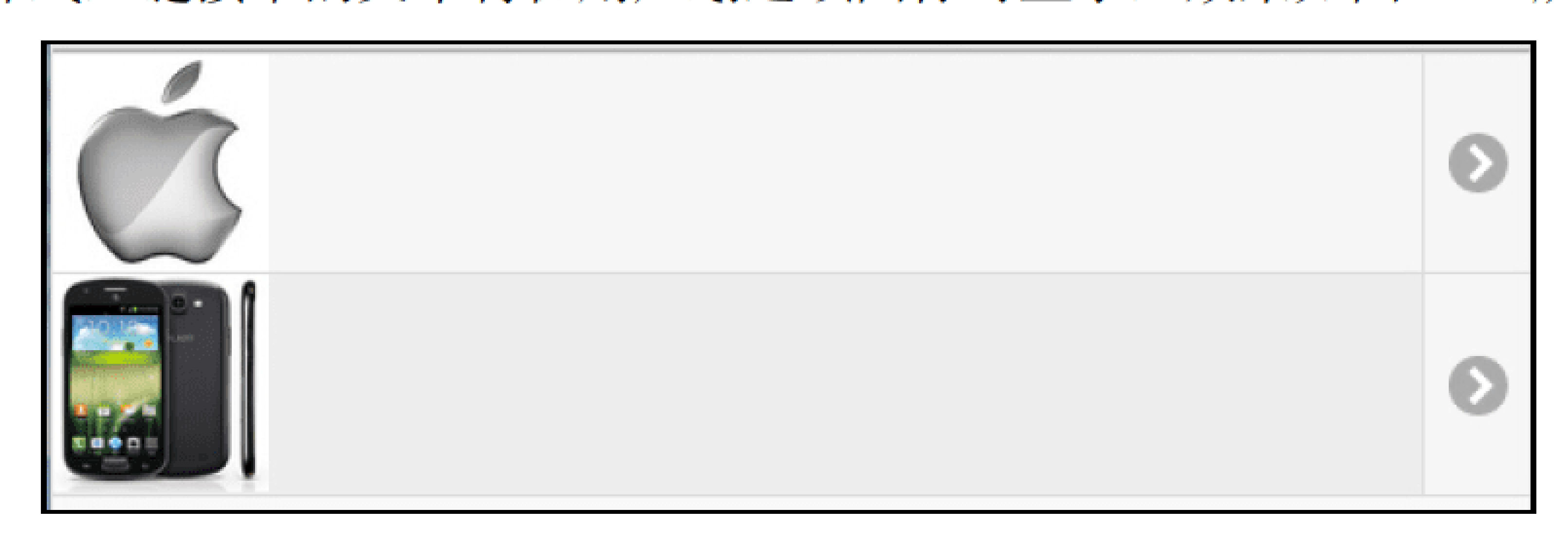


图 9-16　jQuery Mobile 列表内容

jQuery Mobile 的主题提供了许多内置的样式，其中在列表中 ui-li-count 样式类用来指定数字气泡。例如：

```
<ul data-role="listview">
  <li><a href="#">收件箱<span class="ui-li-count">33</span></a></li>
  <li><a href="#">发件箱<span class="ui-li-count">23</span></a></li>
  <li><a href="#">垃圾箱<span class="ui-li-count">7</span></a></li>
</ul>
```

运行效果如图 9-17 所示。

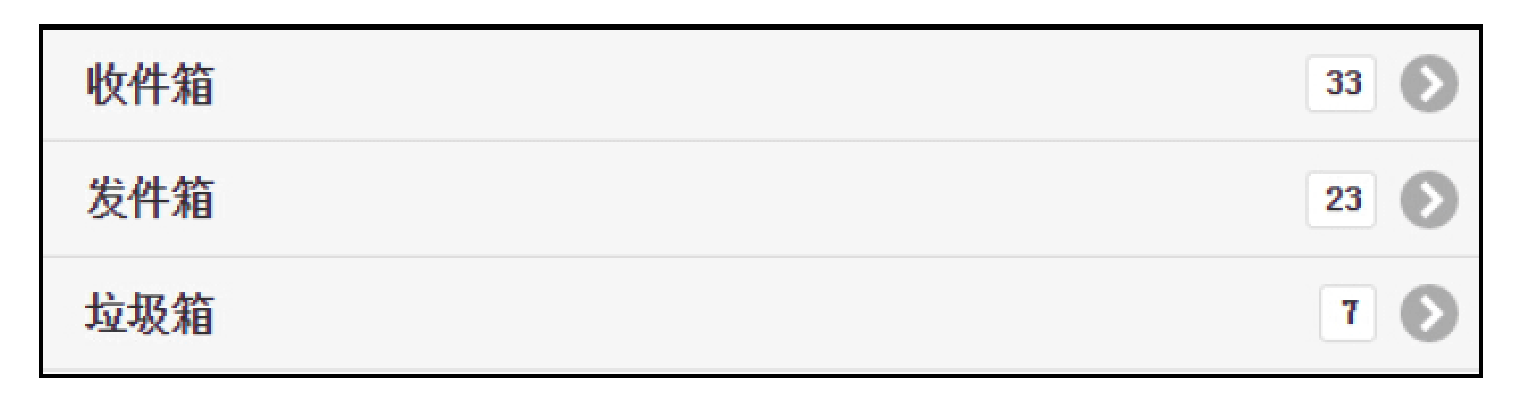


图 9-17　jQuery Mobile 列表样式

9.7　jQuery Mobile 事件

9.7.1　jQuery Mobile 页面事件

jQuery 中存在各种不同类型的页面，这些页面能够相互链接、跳转，就像原生应用的效果一样，这让大多数人感到迷惑。其实 jQuery Mobile 为了达到这样的效果，提供了内置的导航系统，导航系统通过 Ajax 异步加载页面到 DOM 中，并且在显示新页面内容时增加动画效果。将这些链接行为记录在浏览器访问的历史记录中，我们就可以方便地进行页面回退。

在页面显示在屏幕之前，根据时间顺序会触发三个不同的事件：在页面创建前、页面创建、页面初始化。具体见表 9-9。

表 9-9　页面初始化事件

事　　件	描　　述
pagebeforecreate	当页面即将初始化，并且在 jQuery Mobile 已开始增强页面之前，触发该事件
pagecreate	当页面已创建，但增强完成之前，触发该事件
pageinit	当页面已初始化，并且在 jQuery Mobile 已完成页面增强之后，触发该事件

例如：

```
<script>
$(document).on("pageinit",function(){
  alert("触发 pageinit 事件——页面已初始化，DOM 已加载，已完成页面增强。")
});
$(document).on("pagebeforecreate",function(){
  alert("触发 pagebeforecreate 事件——页面即将初始化。仍未开始增强页面。");
});
$(document).on("pagecreate",function(){
  alert("触发 pagecreate 事件——已创建页面，但增强未完成。");
});
</script>
```

同样在通过 Ajax 加载外部页面的时候，也有三个阶段，见表 9-10 所示。

表 9-10　加载外部页面事件

事　　件	描　　述
pagebeforeload	在任何页面加载请求作出之前触发
pageload	在页面已成功加载并插入 DOM 后触发
pageloadfailed	如果页面加载请求失败，则触发该事件。默认情况下将显示"Error Loading Page"消息

例如：

```
<script>
$(document).on("pageload",function(event,data){
  alert("触发 pageload 事件！\nURL: " + data.url);
});
$(document).on("pageloadfailed",function(event,data){
  alert("抱歉，被请求页面不存在。");
});
</script>
<div data-role="page">
  <div data-role="header">
    <h1>欢迎访问我的主页</h1>
  </div>
  <div data-role="content">
    <p>我是一名移动开发者！</p>
     <a href="listview-grid.html">外部页面</a>
  </div>
  <div data-role="footer">
```

```
    <h1>页脚</h1>
  </div>
</div>
```

运行效果如图 9-18 所示。



图 9-18　jQuery Mobile 页面事件

9.7.2　jQuery Mobile 触屏事件

jQuery Mobile 提供了常见的触屏事件处理机制，常见的事件有触摸事件、按住不放事件、向左/右滑动事件，我们可以将事件处理函数绑定到 DOM 元素上。

在快速的一次触摸后会触发 tap 事件，在按住不放后(接近一秒钟的时间)会触发 taphold 事件。例如：

```
<script>
$(document).on("pageinit","#pageone",function(){
  $("#t1 > p").on("tap",function(){
    $(this).hide();
  });
  $("#t2 > p").on("taphold",function(){
    $(this).hide();
  });
});
</script>
<div data-role="content">
    <div id="t1">
      <p>敲击我，我会消失。</p>
      <p>敲击我，我也会消失。</p>
      <p>敲击我，我还会消失。</p>
    </div>
    <div id="t2">
      <p>如果您敲击并保持一秒钟，我会消失。</p>
      <p>敲击并保持住，我会消失。</p>
      <p>敲击并保持住，我也会消失。</p>
    </div>
  </div>
```

swipeleft 事件在用户在某个元素上从左滑动超过 30px 时被触发，swiperight 事件在用户在某个元素上从右滑动超过 30px 时被触发。例如：

```
<body>
 <div role="main" data-role="page" id="pageone">
   <ul data-role="listview" class="touch" data-icon="false"
     data-split-icon="delete">
    <li>
      <a href="#"><img src="_assets/img/apple.png">
        <p class="topic"><strong>apple</strong></p>
      </a>
      <a href="#" class="delete">删除</a>
    </li>
    <li>
      <a href="#"><img src="_assets/img/galaxy_express.png">
       <p class="topic"><strong>galaxy</strong></p>
      </a>
      <a href="#" class="delete">删除</a>
    </li>
  </ul>

<div id="confirm" class="ui-content" data-role="popup" data-theme="a">
       <p id="question">你确定删除:</p>
       <div class="ui-grid-a">
           <div class="ui-block-a">
               <a id="yes" class="ui-btn ui-corner-all ui-mini ui-btn-a"
                 data-rel="back">是</a>
           </div>
           <div class="ui-block-b">
               <a id="cancel" class="ui-btn ui-corner-all ui-mini ui-btn-a"
                 data-rel="back">否</a>
           </div>
       </div>
  </div><!-- /popup -->
  </div>
</body>
<script>
 $(document).on("pageinit","#pageone",function(){
      // 绑定事件
     $( document ).on( "swipeleft swiperight", "#list li", function( event ) {
       var listitem = $( this );
         //判断事件
         if(event.type === "swipeleft"){
          confirmAndDelete( listitem);
         }
      });
      // 如果不是触摸屏设备
      if ( ! $.mobile.support.touch ) {
        // 删除用来在触摸屏设备上隐藏删除按钮的类
        $( "#list" ).removeClass( "touch" );
        // 绑定点击删除事件
        $( ".delete" ).on( "click", function() {
          var listitem = $( this ).parent( "li" );
          confirmAndDelete( listitem );
```

```
        });
    }
    function confirmAndDelete( listitem ) {
      // 高亮显示
      listitem.children( ".ui-btn" ).addClass( "ui-btn-active" );
      // 将 topic 信息插入到 pop 窗口中
      $( "#confirm .topic" ).remove();
      listitem.find( ".topic" ).clone().insertAfter( "#question" );
      // 显示 popup
      $( "#confirm" ).popup( "open" );
      // 绑定确定删除
      $( "#confirm #yes" ).on( "click", function() {
          listitem.remove();
          $( "#list" ).listview( "refresh" );
      });
      // 取消高亮和事件绑定
      $( "#confirm #cancel" ).on( "click", function() {
        listitem.children( ".ui-btn" ).removeClass( "ui-btn-active" );
        $( "#confirm #yes" ).off();
      });
    }
});
</script>
```

显示效果如图 9-19 所示。



图 9-19　jQuery Mobile 触屏事件

9.7.3　jQuery Mobile 方向事件

在用户垂直或水平旋转移动设备时会触发 orientationchange 事件。方向会有两种：portrait 表示竖屏，landscape 表示横屏。

```
<script>
  $(window).on("orientationchange",function(event){
    $("#footer").html("方向改变为：" + event.orientation);
  });
 </script>
<div data-role="page">
  <div data-role="header">
    <h1>orientationchange 事件</h1>
  </div>
  <div data-role="content">
    <p>请试着旋转您的设备！</p>
    <p><b>注释：</b>您必须使用移动设备或者移动模拟器来查看该事件的效果。</p>
  </div>
  <div data-role="footer" id="footer">
    <h1>页脚文本</h1>
  </div>
</div>
```

由于 orientationchange 事件与 window 对象绑定，window.orientation 属性对 portrait 视图返回 0，对 landscape 视图返回 90 或-90。

本章小结

本章学习了 Ajax 的应用场景，使用 Ajax 加载数据完成微博网站中的功能模块，并向服务器提交数据保存到数据库。jQuery Mobile 中使用了大量的 Ajax 技术来完成页面的链接。我们不仅体会到移动端 Web 开发的乐趣，同时还加深了对 jQuery 的理解。

本章练习

1. Ajax 的使用场景有哪些？
2. jQuery Mobile 是什么？
3. jQuery Mobile 有哪些优缺点？

第10章　jQuery动画特效

随着互联网的普及，人们看到的网页特效也越来越多，一些前端技术的出现也使得在网页中添加音效和动画成为可能。jQuery 通过少量的几行代码，就可以实现元素的飞动、淡入淡出等动画效果，还可以自定义各种动画效果。

当我们浏览一个网站的时候，网站给人的第一印象就是网站的外观。当我们在电子商务网站上购物时，网站的互动型至关重要，操作流程过多、用户找不到想要的导航、广告太多等都会导致用户离开网页。UEO 是英文 User Experience Optimization 的缩写，把它翻译成中文的意思是用户体验优化，也就是网站针对用户体验来进行优化，让用户感觉到网站非常好。如何能最大程度地优化页面的用户体验度，是每个前端页面开发人员在设计页面时需要考虑的一个重要问题。无可置疑，jQuery 中众多的动画与特效为提高页面的用户体验度带来了极大的方便。

本章内容：

- 掌握 jQuery 动画特效

10.1　jQuery 常用特效

10.1.1　显示和隐藏特效

jQuery 能够立即显示或者隐藏 DOM 元素，使用的方法分别为 show 和 hide 方法。例如：

```
$( 'p' ).hide();
$('hidden').show()
```

在传统的 JavaScript 中，需要改变元素的 display 属性：当隐藏元素时会设置元素的 display 属性为 block；当显示元素时会设置元素的 display 属性为 none，此时元素的宽度和高度都为 0，而不是把元素变为透明并在页面上留下一片空的区域。

jQuery 的显示和隐藏也是通过设置元素的 display 属性来实现的，并且我们可以在显示和隐藏元素的时候使用动画特效。只需要在 show 方法中传入 slow、normal 或 fast 参数即可。show 方法的语法如下所示：

```
show(duration ,[callback])
```

duration 参数可以是一个字符串或者数字，用来表示动画执行的速度。字符串的取值可以是 slow、normal、fast，其对应的速度分别是 0.6 秒、0.4 秒、0.2 秒，默认值为 0.4 秒。如

果我们想更精确地控制动画执行的时间，可以传入一个数字，比如 500，表示该动画执行的时间为 500 毫秒。callback 可选参数是动画执行后立即执行的回调函数。

hide 方法的语法与 show 方法类似，参数也都相同，如下所示：

```
hide(duration ,[callback])
```

例如：

```
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
 <style type="text/css">
 p {
    background: yellow;
  }
</style>
 </head>
 <body>
<button id="show">显示</button>
<button id="hide">隐藏</button>
<p style="display: none">Hello  2</p>
 <script type="text/javascript">
$( "#show" ).click(function() {
  $( "p" ).show( "slow" );
});
$( "#hide" ).click(function() {
  $( "p" ).hide( 3000 ,function() {
    alert("动画完成.");
  });
});
</script>
 </body>
</html>
```

运行效果如图 10-1 所示。

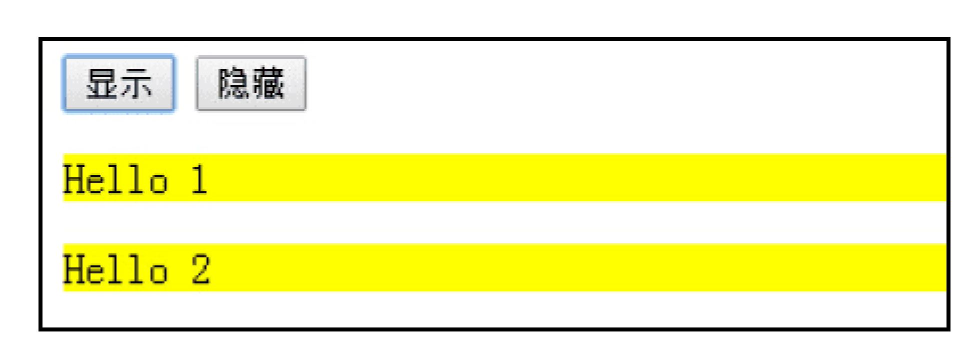


图 10-1　显示和隐藏特效

当隐藏元素时，通常也需要再把元素显示在页面上。为了实现切换显示状态，在代码中需要检测当前元素的显示状态，然后根据该状态再执行元素是显示还是隐藏。这多少有些繁琐，为此，jQuery 提供了 toggle()方法，该方法的功能就是切换元素的可见状态，即如果是显示状态，则变成隐藏状态；如果是隐藏状态，则变成显示状态。例如：

```
$( "p" ).toggle();
```

```
$( "p" ).toggle( "slow" );
$( "p" ).toggle( 1800 );
```

那么上面使用 hide 和 show 方法的例子，可以修改为使用 toggle 方法，代码如下：

```
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
 <style type="text/css">
 p {
    background: yellow;
  }
</style>
 </head>
 <body>
<button>切换</button>
<p style="display: none">Hello  1</p>
<p style="display: none">Hello  2</p>
 <script type="text/javascript">
$( "button" ).click(function() {
  $( "p" ).toggle( "slow",function() {
    alert("动画完成.");
  } );
});
</script>
 </body>
</html>
```

10.1.2　淡入淡出特效

在桌面系统和 PPT 中淡入淡出效果比较常见，jQuery 可以方便地为网页提供淡入、淡出效果。jQuery 中对应淡入效果的方法是 fadeIn 方法，对应淡出效果的方法是 fadeTo 方法。与 show()和 hide()方法比较，fadeIn 和 fadeOut 方法不是改变元素的 width 与 height 属性，而是改变元素的透明度。

fadeIn 方法的语法为：

```
fadeIn ( [duration ] [,callback ] )
```

duration 参数可以是一个字符串或者数字，用来表示动画执行的速度。字符串的取值可以是 slow、normal、fast，其对应的速度分别是 0.6 秒、0.4 秒、0.2 秒，默认值为 0.4 秒。如果我们想更精确地控制淡入特效的执行时间，可以传入一个数字。

fadeOut 方法的语法为：

```
fadeOut ( [duration ] [,callback ] )
```

例如：

```
$("#div1").fadeIn();
```

```
$("#div2").fadeIn("slow");
$("#div3").fadeOut(3000);
```

下面我们来实现一个图片淡入淡出的案例，效果如图 10-2 所示。



图 10-2　淡入淡出特效

通过淡入淡出效果显示不同的图片，并且当显示到最后一张图片时自动循环从第一张图片显示。如果要实现这个功能，除了用到 fadeIn 和 fadeOut 方法外，还要使用 jQuery 选择器选择下一张图片(或者第一张图片)，为了不间断地循环执行，还会使用到 setInterval 函数。具体实现的代码如下所示：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<style type="text/css">
header{
background:#fff  ;
float:right;
position:relative; /* 相对定位，作为绝对定位的 img 的父级 */
}
header img {
vertical-align:top; /** 兼容 IE6，去掉 IE6 中 img 下面有空隙 **/
position:absolute; /** 绝对定位，所有的图片都显示在同一个位置 **/
right:0;
}
</style>
</head>
<body>
    <header>
        <img alt="布宜诺斯艾利斯" src="_assets/img/buenosaires.jpg">
        <img alt="开普敦" src="_assets/img/capetown.jpg">
        <img alt="纽约" src="_assets/img/newyork.jpg">
    <header>
```

```
</body>
<script>
$(function(){
    //先隐藏所有图片，再将对象移到第一张图片，使之淡入
    $("header img").fadeOut(0).eq(0).fadeIn(0);
    //初始化图片的索引值
    var i = 0;
    //setInterval 是每隔一段时间循环一个动作
    setInterval(function(){
        //判断是否执行到最后一张图片
        if($("header img").length > (i+1)){
            //索引值为 i 的图片淡出，它的下一张图片淡入
            $("header img").eq(i).fadeOut(2000).next("img").fadeIn(2000);
            //使索引值增加 1，下一次执行动作的图片索引值为 i+1
            i++;
        }
        else{//如果为最后一张图片，将执行下面的代码
        //索引值为 i 的图片淡出，这里不同于上面，是第一张图片淡入
        $("header img").eq(i).fadeOut(2000).siblings("img").eq(0).fadeIn(2000);
            i = 0;//将索引值变为 0，回到初始状态
        }
    },3000);//3000ms 执行一次淡入淡出的动作
})
</script>
</html>
```

上面的代码使用$("header img")选定对应的 img 数组；通过 eq(i)选择数组中的一张图片；fadeOut(2000)设定时间为 2 秒的淡出效果；使用 next("img")改变对象，将操作对象移到下一张图片；fadeIn(2000)设定时间为 2 秒的淡入效果。同时我们使用链式操作写在一行：

```
$("header img").eq(i).fadeOut(2000).next("img").fadeIn(2000)
```

jQuery 还提供了 fadeTo 方法以调整元素的透明度，并且以动画的效果调整到指定的不透明度值。因为 fadeIn 和 fadeOut 方法只能将元素的透明度设置为 1 或 0，所以如果想制作半透明特效，应该使用 fadeTo 方法。例如：

```
$( "p" ).fadeTo( "slow", 0.33 );
```

fadeTo 方法的语法为：

```
fadeTo(duration,opacity,[callback])
```

参数 speed 为动画效果的速度，参数 opacity 为指定的不透明值，取值范围是 0.0 到 1.0，可选项参数 callback 为动画完成时执行的函数。

10.1.3 滑动特效

我们注意到在 show 和 hide 方法特效执行的时候，实际上是同时改变元素的高度和宽度。如果我们只想使用改变高度的效果来显示元素，就应该使用 jQuery 提供的 slideDown 和

slideUp 方法。

slideDown 方法以动画的效果将所选择元素的高度向下滑动增加，它的语法如下：

```
slideDown(duration,[callback])
```

slideUp 方法以动画的效果将所选择元素的高度向上滑动减少，它的语法如下：

```
slideUp (duration,[callback])
```

例如：

```
<html>
<head>
 <meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
 <style type="text/css">
    .slideDownbox, .slideUpbox, .slideTogglebox{
        float:left;
        padding:8px;
        margin:16px;
        border:1px solid red;
        width:200px;
        height:50px;
        background-color:#000000;
        color:white;
    }
    .clear{
        clear:both;
    }
</style>
</head>
<body>
<h1>jQuery slideUp(), slideDown()</h1>
<div class="clear">
    <h2>slideDown() example</h2>
    <div class="slideDownbox">
        Click me - slideDown()
    </div>
    <div class="slideDownbox">
        Click me - slideDown()
    </div>
</div>
<div class="clear">
    <h2>slideUp() example</h2>
    <div class="slideUpbox">
        Click me - slideUp()
    </div>
    <div class="slideUpbox">
        Click me - slideUp()
    </div>
</div>
```

```
<br/><br/>
<div class="clear">
    <button id="reset">Reset</button>
</div>
<script type="text/javascript">
$(".slideDownbox").click(function () {
   $(this).hide().slideDown('slow');
});
$(".slideUpbox").click(function () {
   $(this).slideUp(2000);
});
$("#reset").click(function(){
    location.reload();
});
</script>
</body>
</html>
```

运行效果如图 10-3 所示。

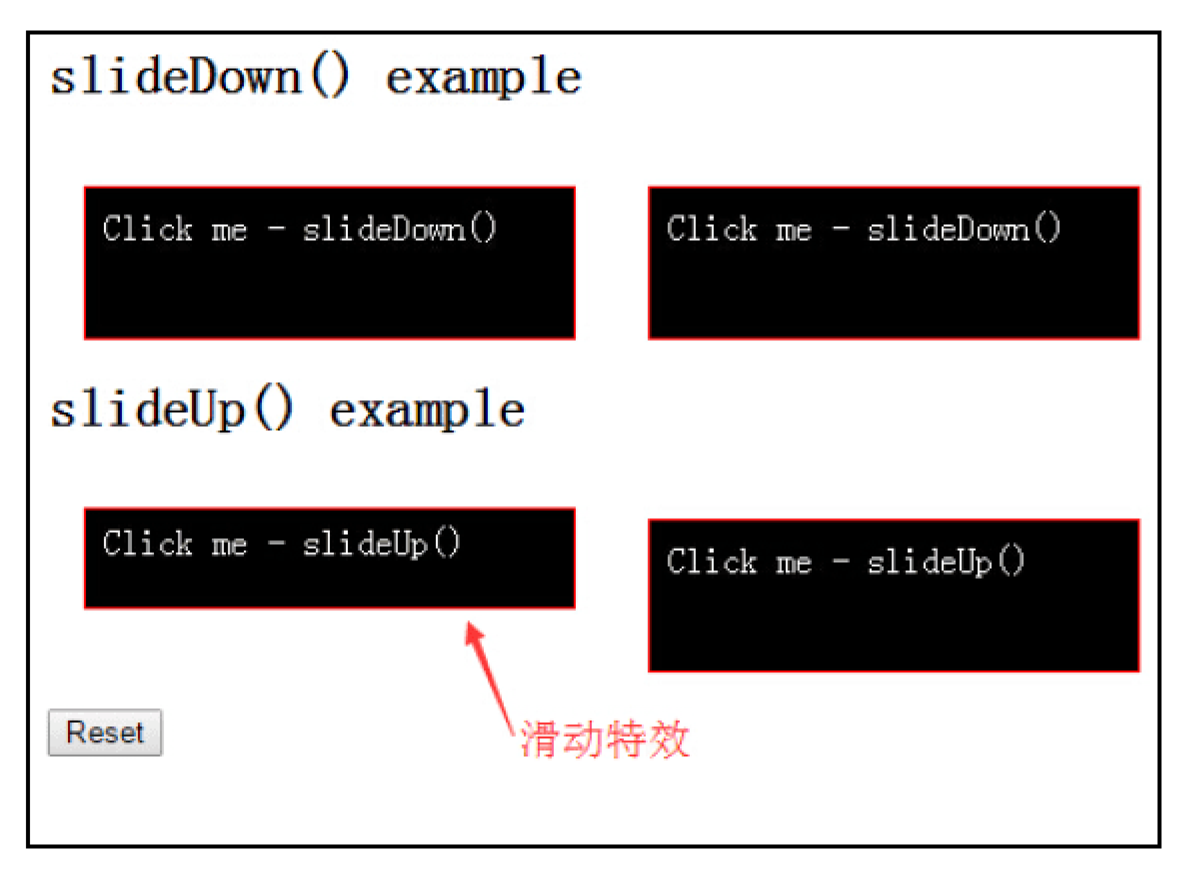


图 10-3　滑动特效

像 toggle 方法能够自动切换显示状态一样，jQuery 提供的 slideToggle 方法也能够根据当前元素的显示状态，使用滑动效果自动进行切换。它的语法为：

```
slideToggle(duration,[callback])
```

例如：

```
<html>
<head>
 <meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
 <style type="text/css">
    .slideDownbox, .slideUpbox, .slideTogglebox{
        float:left;
        padding:8px;
        margin:16px;
        border:1px solid red;
```

```
            width:200px;
            height:50px;
            background-color:#000000;
            color:white;
        }
        .clear{
            clear:both;
        }
    </style>
    </head>
    <body>
    <h1>slideToggle() example</h1>
    <div class="clear">
        <h2>slideToggle() example</h2><button id=slideToggle>slideToggle()</button>
        <br/>
        <div class="slideTogglebox">
            slideToggle()
        </div>
        <div class="slideTogglebox">
            slideToggle()
        </div>
    </div>
    <br/><br/>
    <div class="clear">
        <button id="reset">Reset</button>
    </div>
    <script type="text/javascript">
    $("#slideToggle").click(function () {
      $('.slideTogglebox').slideToggle();
    });
     $("#reset").click(function(){
        location.reload();
    });
    </script>
    </body>
    </html>
```

运行效果如图 10-4 所示。

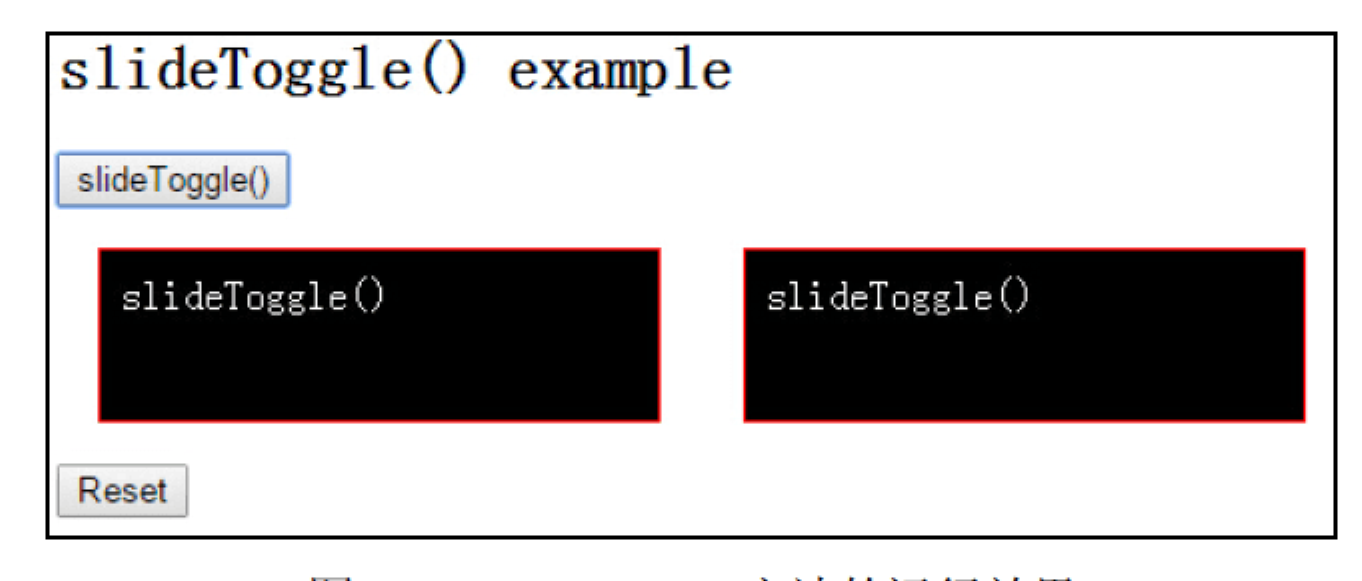


图 10-4　slideToggle 方法的运行效果

10.2　创建自定义动画

10.2.1　基本自定义动画

jQuery 还提供了更灵活的创建动画的方式，那就是自定义动画。通过使用 animate 方法，我们可以设置动画运行的一些高级属性，也就能创建出更复杂、更优雅的动画了。

animate 方法的语法格式如下：

```
animate( properties, [ duration ], [ easing ], [ callback ] )
```

其中，参数 properties 表示动画改变的目的属性样式和值的集合。可选参数 duration 表示三种默认的速度(slow、normal、fast)或自定义的数字。可选参数 easing 用于控制动画的表现效果，通常有 linear 和 swing 字符值。可选参数 callback 为动画完成后执行的回调函数。我们前面学习的动画特效都是基于此函数实现的。

比如，我们使用 animate 方法来实现改变宽度、透明度的动画效果，代码如下：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<title>animate</title>
<style type="text/css">#content{background-color:#fa0;width:300px;
    height:30px;padding:3px}</style>
</head>
<body>
<input type="button" id="animate" value="Animate">
<div id="content">Animate Height</div>
<script>
$("#animate").click(function() {
    $("#content").animate(
            {"height": "80px"},
            "fast");
});
$("#animate").click(function() {
    $("#content").animate(
            {"opacity": "0.15"},
            "slow");
});
</script>
</body>
</html>
```

运行效果如图 10-5 所示。

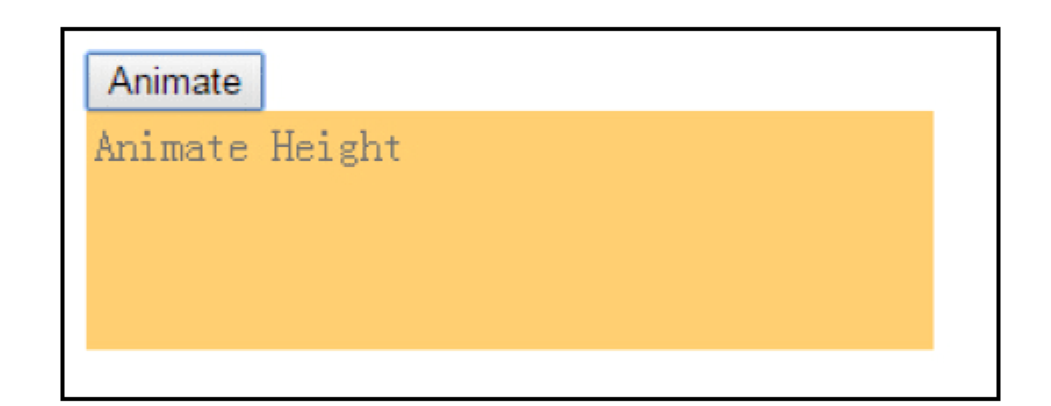


图 10-5　animate 方法的运行效果

通过使用 animate 方法，我们可以移动元素的位置，例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<title>animate</title>
<style>#content{background-color:#68f;position:absolute;width:100px;height:
    100px;padding:3px;margin-top:5px;left:100px}</style>
</head>
<body>
<input type="button" id="left" value="Left">
<input type="button" id="right" value="Right">
<div id="content">Move</div>
<script>
$("#right").click(function() {
   $("#content").animate(
          {"left": "+=150px"},
          "slow");
});
$("#left").click(function() {
   $("#content").animate(
          {"left": "-=150px"},
          "slow");
});
</script>
</body>
 </html>
```

在上面的代码中，设置元素目标位置时使用了表达式，点击按钮后方块会移动 150 个像素，运行效果如图 10-6 所示。

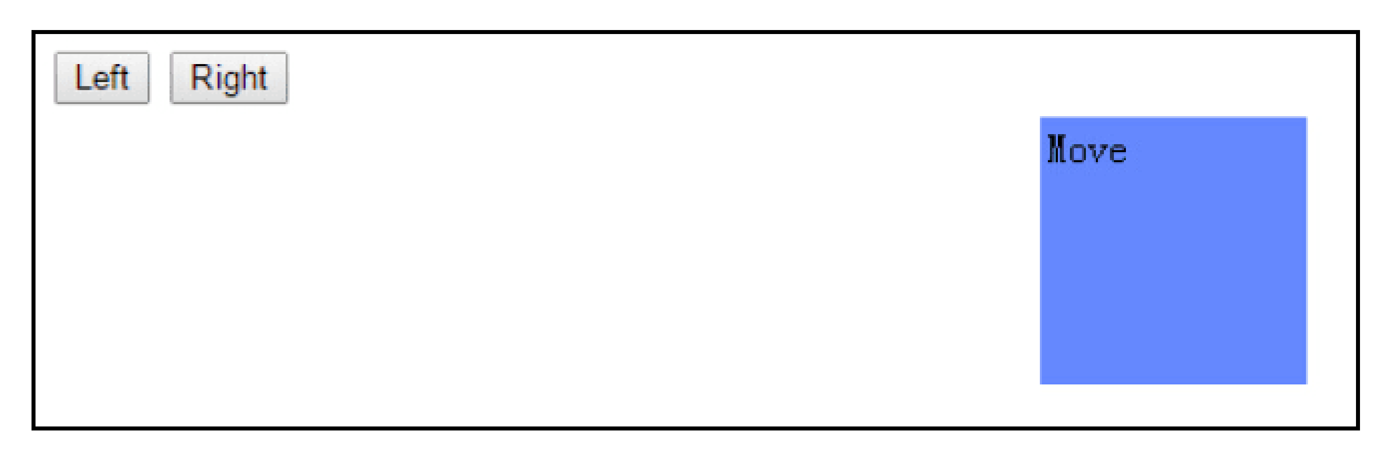


图 10-6　animate 使用了表达式

10.2.2　自定义动画的回调函数

同样地，在 animate 方法中我们也可以设置回调函数，以记录动画执行完成，并执行后续的操作。例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<title>animate</title>
<style type="text/css">#content{background-color:#fa0;width:300px;
    height:30px;padding:3px}</style>
</head>
<body>
<input type="button" id="animate" value="Animate">
<div id="content">Animate Height</div>
<script>
$("#animate").click(function() {
   $("#content").animate(
         {"height": "100px", "width": "250px"},
         "slow", function(){
            $(this).html("Animation Completed");
         });
});
</script>
</body>
</html>
```

运行效果如图 10-7 所示。

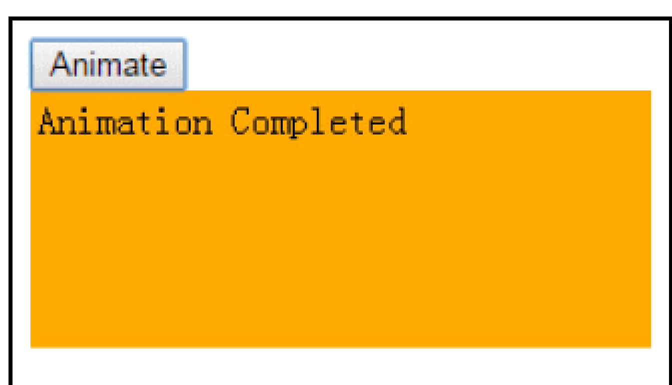


图 10-7　animate 回调

10.2.3　动画队列

只使用一个动画效果制作不出新颖的动画，复杂的动画通常都需要由多个动画效果组成。如何控制一个元素执行多个动画效果呢？jQuery 使用队列的概念对多个动画效果进行排队，形成“队列”，执行的时候也按照“队列”的顺序播放动画。jQuery 定义了全局的 queue、dequeue 函数，在 jQuery 对象上则定义了 queue、dequeue、delay、clearQueue 等方法。

jQuery.queue 的语法格式如下：

```
jQuery.queue( element [, queueName ] [, callback ])
```

element 是 DOM 元素对象，queueName 表示队列的名称，默认是 fx。callback 是添加到队列的函数或函数数组。如果 callback 对象为函数，则将此函数追加到队列中；如果 callback 对象为函数数组，则将使用此数组替换队列中的内容。如果省略 callback 参数，jQuery.queue 方法会返回队列中指定元素的回调函数数组。例如：

```
var element  = $( "div" )[ 0 ];
function cb1() {alert(1)}
function cb2() {alert(2)}
function cb3() {alert(3)}
var arr = [cb3, cb2];
$.queue(element , 'foo', cb1); // 第三个参数为 function
$.queue(element , 'foo', arr); // 第三个参数为数组
console.log($.queue(element , 'foo'));
//输出[function, function]0: function cb3() {alert(3)}1: function cb2() {alert(2)}
```

jQuery.dequeue 的语法格式如下：

```
jQuery.dequeue( element [, queueName ] )
```

比如在上面的代码基础上，我们执行下面的代码：

```
$.dequeue(element , 'foo');
```

将会得到如图 10-8 所示的效果。

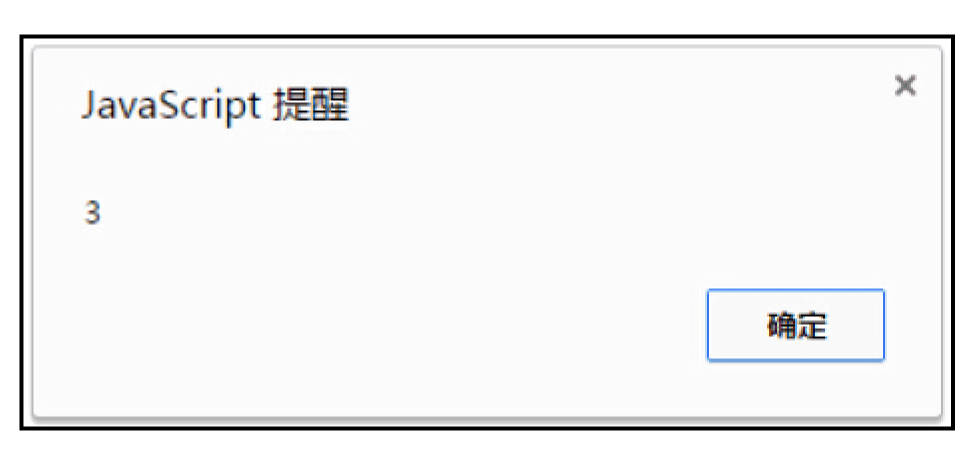


图 10-8　dequeue 的效果

可见，如果不执行 dequeue，那么队列中的下一个函数永远不会执行。

相比之下，在 jQuery 对象上调用 queue 和 dequeue 方法更为常用，在 jQuery 对象上调用 queue 方法和 dequeue 方法的语法格式如下：

```
queue( [queueName ][, callback] )
dequeue( [queueName ] )
```

它们的参数与对应的全局函数的参数说明一致。

例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<title>animate</title>
<style type="text/css">
```

```
div {
    margin: 3px;
    width: 40px;
    height: 40px;
    position: absolute;
    left: 0px;
    top: 30px;
    background: green;
    display: none;
  }
  div.newcolor {
    background: blue;
  }</style>
</head>
<body>
<p>队列的长度是: <span></span></p>
<div></div>
<script>
var div = $( "div" );
function runIt() {
    div.show( "slow" )
    .animate({ left: "+=200" }, 2000 )
    .slideToggle( 1000 )
    .slideToggle( "fast" )
    .animate({ left: "-=200" }, 1500 )
    .hide( "slow" )
    .show( 1200 )
    .slideUp( "normal", runIt );
}
function showIt() {
  var n = div.queue( "fx" );
  $( "span" ).text( n.length );
  setTimeout( showIt, 100 );
}
runIt();
showIt();
</script>
</body>
</html>
```

在 runIt()函数中，我们在 div 元素上添加了许多动画效果，也就是在队列中增加了许多回调函数，当最后一个动画执行完毕的时候再一次调用 runIt()函数，这样整个动画效果会循环执行。同时用 showIt()函数检测当前队列中待执行的回调函数的个数。运行效果如图 10-9 所示。

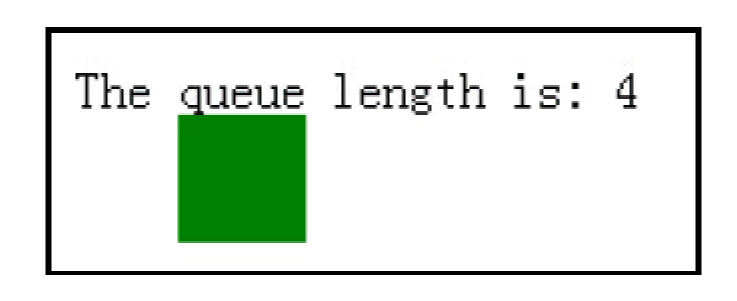


图 10-9　检测队列中动画的个数

有时需要停止执行队列总剩余的动作或动画，我们可以调用 clearQueue()方法清空队列。例如：

```
$( "div" )
    .queue( "fx", function( next ) {
        console.log( "在 fx 队列中" );
    } )
    .clearQueue( "fx" )
    .dequeue( "fx" );
```

上面的代码不会执行输出语句，因为已经清空队列了。

还可以将队列中的函数延时执行，delay()函数的语法格式如下所示：

```
delay(duration [,queueName])
```

delay()函数设置一个延时来推迟执行队列中之后的动画。例如：

```
$( "div " ).slideUp( 300 ).delay( 500 ).fadeIn( 400 );
```

上面的代码在向上滑动效果和淡入效果之间会暂停 500 毫秒。

10.2.4　停止特效

jQuery 提供了 stop()函数用于停止特效。语法为：

```
stop( [queue ] [, clearQueue ] [, jumpToEnd ] )
```

当在 DOM 元素上调用此方法时，当前正在执行的动画会立即停止，回调函数也不会触发。当 DOM 元素中有多个动画方法被调用时，下一个动画方法会立即执行。

第一个参数表示队列名。

第二个参数如果指定为 true，则不仅停止当前动画，而且停止在动画队列中等待执行的所有其他的动画。

第三个参数如果指定为 true，则会直接执行到当前动画的结束状态，比如.slideUp()动画执行的时候调用 stop，那么元素会马上隐藏，回调函数也会马上执行。

例如：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<script src="http://apps.bdimg.com/libs/jQuery/2.1.1/jQuery.js"></script>
<title>animate</title>
<style type="text/css">
div {
    margin: 3px;
    width: 40px;
    height: 40px;
    position: absolute;
    left: 0px;
```

```
    top: 30px;
    background: green;
    display: none;
  }
  div.newcolor {
    background: blue;
  }</style>
</head>
<body>
<button id="start">Start</button>
<button id="stop1">Stop()</button>
<button id="stop2">Stop(true)</button>
<button id="stop3">Stop(true,true)</button>
<div></div>
<script>
$( "#start" ).click(function() {
  $( "div" )
    .show( "slow" )
    .animate({ left: "+=200" }, 5000 )
    .queue(function() {
      $( this ).addClass( "newcolor" ).dequeue();
    })
    .animate({ left: '-=200' }, 1500 )
    .queue(function() {
      $( this ).removeClass( "newcolor" ).dequeue();
    })
    .slideUp();
});
$( "#stop1" ).click(function() {
  $( "div" )
    .stop();
});
$( "#stop2" ).click(function() {
  $( "div" )
    .stop(true);
});
$( "#stop3" ).click(function() {
  $( "div" )
    .stop(true,true);
});
</script>
</body>
</html>
```

效果如图 10-10 所示。

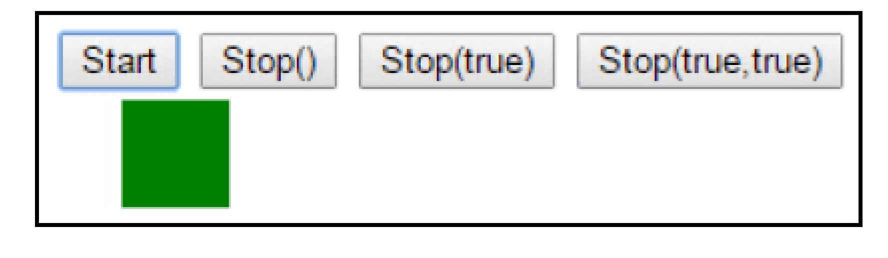


图 10-10　停止特效

在上面的代码中，点击 Start 按钮后，会再添加多个动画特效，这里还结合了改变 div 元

素的颜色，使用了下面的代码：

```
.queue(function() {
     $( this ).addClass( "newcolor" ).dequeue();
 })
```

在队列中加入函数处理，而且在函数处理中调用了 dequeue()方法，这样就会调用队列中后一个动画特效(函数)，形成一段连贯的动画。整个效果就是首先显示 div 元素，然后向右动 200 像素，然后改变 div 元素颜色为蓝色，然后向左移动 200 像素，继续改变 div 元素颜色为绿色，最后滑动隐藏 div 元素。我们增加了三个按钮，分别调用 stop 函数停止动画，但是使用的参数不同，得到的停止动画的效果也不同。第一个 Stop 按钮只会停止当前动画，第二个 Stop 按钮则会停止当前动画和队列中的所有动画，第三个按钮则会直接执行到当前动画的结束状态并不执行队列中的后续动画。

在人人微博网站中，我们可以为微博文章设置淡入淡出效果或者其他效果。

jQuery 前端动画的原理是通过一个定时器 setInterval 每隔一定时间来改变元素的样式，动画结束时使用 clearInterval 即可。jQuery 动画虽然易用高效，但是性能上存在问题。一般需要结合其他的动画实现方式一起使用，如 CSS3 中对动画的支持。

10.3　综合案例赏析

下面我们来模仿实现手机端的一些动画特效，在 Chrome 浏览器中打开页面，如图 10-11 所示。



图 10-11　案例效果图

当我们单击左上角的按钮的时候，内容会滑动到屏幕右侧并缩小显示，同时一个导航列表会显示在屏幕左侧，最上方的状态条也会改变。

当单击导航或内容时，也会以动画效果隐藏导航栏并显示内容页。我们使用的图片资源

如图 10-12 所示。

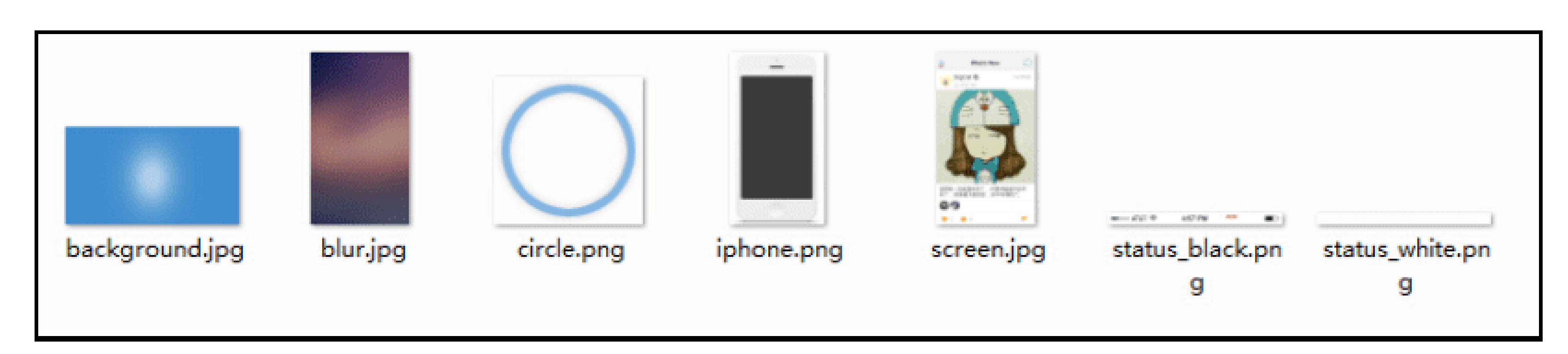


图 10-12　案例资源

HTM5 网页代码如下：

```
<html>
<head>
    <title></title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
        maximum-scale=1, user-scalable=no">
    <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
    <div class="background">
        <div class="iphone">
            <div class="screen">
                <div class="menu">
                    <ul>
                        <li><a href="">Home</a></li>
                        <li><a href="">What's New</a></li>
                        <li><a href="">流行</a></li>
                        <li><a href="">活动</a></li>
                        <li><a href="">聊天</a></li>
                        <li><a href="">周围</a></li>
                        <li><a href="">帮助</a></li>
                        <li><a href="">登出</a></li>
                    </ul>
                </div>
                <div class="status"></div>
                <div class="content ">
                    <div class="button"></div>
                </div>
            </div>
        </div>
    </div>
    <script src="http://code.jQuery.com/jQuery-latest.min.js"></script>
    <script src="menu.js"></script>
</body>
</html>
```

在内容区域我们使用 screen.jpg 作为背景图片，增加<div class="button"></div>作为按钮区域。通过分析，我们需要两个动画效果：我们命名“动画 1”为内容页的隐藏和导航栏的显示，命名“动画 2”为导航栏的隐藏和内容页的显示。具体见样式文件，代码如下：

```
/*  基本的复位和通用 */
    * {
        margin: 0px;
        padding: 0px;
    }
    html, body {
        min-width: 435px;
        min-height: 1000px;
        -webkit-touch-callout: none;
        -webkit-user-select: none;
        -khtml-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
    }
    div.background {
        position: absolute;
        top: 0px; left: 0px;
        min-width: 435px;
        min-height: 1000px;
        height: 100%; width: 100%;
        background: #408dce url('img/background.jpg') no-repeat center center;
        -webkit-background-size: cover;
           -moz-background-size: cover;
             -o-background-size: cover;
                background-size: cover;
    }
/*  设备 */
    div.iphone {
        height: 810px; width: 435px;
        position: absolute;
        margin: auto;
        top: 0px; bottom: 80px; left: 0px; right: 0px;
        background: url('img/iphone.png') no-repeat center center;
        -webkit-background-size: cover;
           -moz-background-size: cover;
             -o-background-size: cover;
                background-size: cover;
    }
    div.screen {
        height: 568px; width: 320px;
        position: absolute;
        top: 119px; left: 59px;
        overflow: hidden;
        position: relative;
    }
/*  内容 */
    div.menu {
        position: absolute;
        height: 100%; width: 100%;
        background: url('img/blur.jpg') no-repeat center center;
```

```
    -webkit-background-size: cover;
       -moz-background-size: cover;
         -o-background-size: cover;
            background-size: cover;
}
    div.menu ul {
        width: 600px;
        list-style: none;
        margin: 95px 0px 0px 75px;
    }
    div.menu li {
        font-weight: 200;
        font-size: 19px;
        line-height: 19px;
        margin-bottom: 28px;
        /*  隐藏导航  */
        -webkit-transition: all 400ms ease-in-out, opacity 1000ms ease;
           -moz-transition: all 400ms ease-in-out, opacity 1000ms ease;
            -ms-transition: all 400ms ease-in-out, opacity 1000ms ease;
             -o-transition: all 400ms ease-in-out, opacity 1000ms ease;
                transition: all 400ms ease-in-out, opacity 1000ms ease;
        -webkit-transform: scale(0.2);
           -moz-transform: scale(0.2);
                transform: scale(0.2);
        opacity: 0;
    }
    div.menu li.visible {
        /*  显示导航  */
        -webkit-transform: scale(1);
           -moz-transform: scale(1);
                transform: scale(1);
        opacity: 1;
    }
    div.menu li a {
        color: rgba(255,255,255,0.9);
        text-decoration: none;
    }
div.status {
    position: absolute;
    top: 0px; left: 0px;
    z-index: 200;
    height: 20px; width: 100%;
    background: url('img/status_black.png') no-repeat center center;
    -webkit-background-size: cover;
       -moz-background-size: cover;
         -o-background-size: cover;
            background-size: cover;
}
div.status.active {
    background-image: url('img/status_white.png');
}
```

```
div.content {
    position: absolute;
    top: 0px; left: 0px;
    height: 568px; width: 320px;
    background: #fff url('img/screen.jpg') no-repeat center center;
    -webkit-background-size: cover;
       -moz-background-size: cover;
         -o-background-size: cover;
            background-size: cover;
    -webkit-transition: all 300ms ease-in-out;
       -moz-transition: all 300ms ease-in-out;
        -ms-transition: all 300ms ease-in-out;
         -o-transition: all 300ms ease-in-out;
            transition: all 300ms ease-in-out;
    -webkit-backface-visibility: hidden;
}
    /*  隐藏内容页  */
    div.content.inactive {
        cursor: pointer;
        -webkit-transform: translate(160px, 0px) scale(0.5);
           -moz-transform: translate(160px, 0px) scale(0.5);
            -ms-transform: translate(160px, 0px) scale(0.5);
                transform: translate(160px, 0px) scale(0.5);
    }
    div.button {
        width: 30px; height: 30px;
        position: absolute;
        top: 23px; left: 3px;
        cursor: pointer;
    }
```

在上面的代码中，我们使用-webkit-transform:scale(0.2)隐藏导航，使用-webkit-transform: scale(1)显示导航。div.content.inactive 类的内容为隐藏内容页的动画效果。

JavaScript 代码如下：

```
/*  显示导航 */
$('div.button').on('click', function(){
    //检测内容页是否在前面
    if( !$('div.content').hasClass('inactive') ){
        // 滑动并缩小内容页
        $('div.content').addClass('inactive');
        setTimeout(function(){ $('div.content').addClass('flag'); }, 100);

        // 改变手机状态条为黑色
        $('div.status').fadeOut(100, function(){
            $(this).toggleClass('active').fadeIn(300);
        });
        // 有时间差的为导航中选项添加动画效果
        var timer = 0;
        $.each($('li'), function(i,v){
```

```
                timer = 40 * i;
                setTimeout(function(){
                    $(v).addClass('visible');
                }, timer);
            });
        }
    });
    /*  隐藏导航并显示内容页 */
    function closeMenu() {
        // 以动画效果显示内容
        $('div.content').removeClass('inactive flag');
        // 改变状态条为白色
        $('div.status').fadeOut(100, function(){
            $(this).toggleClass('active').fadeIn(300);
        });
        //隐藏导航
        setTimeout(function(){
            $('li').removeClass('visible');
        }, 300);
    }
    //内容点击
    $('div.content').on('click', function(){
        if( $('div.content').hasClass('flag') ){
            closeMenu();
        }
    });
    //导航点击
    $('li a').on('click', function(e){
        e.preventDefault();
        closeMenu();
    });
```

在上面的代码中添加了单击内容页返回按钮的事件、单击内容页的事件、单击导航的事件。内容页返回按钮的事件会触发“动画 1”效果，单击内容页的事件和单击导航的事件会触发“动画 2”效果。为了在内容页显示在最前面时单击内容页的事件不执行动画，我们添加了 flag 类作为标识以区别。至此我们就完成了既好看又实用的动画效果。

本章小结

本章介绍了 jQuery 对动画特效的支持，从立即显示隐藏元素到有动画效果地显示隐藏元素，还介绍了允许创建自定义动画的 animate 方法，你还了解到 jQuery 的队列函数以及如何将动画排队以串行方式执行。

本章练习

1. 使用 jQuery 如何实现动画特效？
2. 请介绍 jQuery 中的队列函数。

附录A　服务端接口

A.1　新增用户

接口说明：新增用户。

URL：http://localhost:8080/rrwb/crud/user/add。

HTTP 请求方式：POST/GET。

请求参数：如表 A-1 所示。

表 A-1　新增用户的请求参数

参 数 名 称	类　型	说　明
login_name	String	登录名，必输
nick_name	String	昵称，必输
img	String	缩略图地址
real_name	String	真实姓名
provinceValue	String	所在省份编号
provinceName	String	所在省份名称
cityValue	String	所在城市编号
cityName	String	所在城市名称
gender	String	性别
age	String	年龄
other_email	String	备用邮箱
blog	String	博客地址
qq	String	QQ
pwd	String	密码，必输
msn	String	MSN
birthday	String	生日

响应参数：如表 A-2 所示。

表 A-2　新增用户的响应参数

参数名称	类　型	说　明
code	Number	0：插入失败 1：插入成功

A.2　编辑用户个人信息

接口说明：编辑用户个人信息。

URL：http://localhost:8080/rrwb/crud/user/edit。

HTTP 请求方式：POST/GET。

请求参数：如表 A-3 所示。

表 A-3　编辑用户个人信息的请求参数

参数名称	类　型	说　明
login_name	String	登录名，必输
nick_name	String	昵称，必输
img	String	缩略图地址
real_name	String	真实姓名
provinceValue	String	所在省份编号
provinceName	String	所在省份名称
cityValue	String	所在城市编号
cityName	String	所在城市名称
gender	String	性别
age	String	年龄
other_email	String	备用邮箱
blog	String	博客地址
qq	String	QQ
pwd	String	密码，必输
msn	String	MSN
birthday	String	生日

响应参数：如表 A-4 所示。

表 A-4　编辑用户个人信息的响应参数

参数名称	类　型	说　明
code	Number	0：插入失败 1：插入成功

A.3　根据用户 ID 查询用户

接口说明：根据用户 ID 查询用户。

URL：http://localhost:8080/rrwb/crud/user/edit。

HTTP 请求方式：POST/GET。

请求参数：如表 A-5 所示。

表 A-5　根据用户 ID 查询用户的请求参数

参 数 名 称	类　　型	说　　明
_id	Number	用户 ID，必输

响应参数：如表 A-6 所示。

表 A-6　根据用户 ID 查询用户的响应参数

参 数 名 称	类　　型	说　　明
code	Number	0：插入失败 1：插入成功

A.4　根据用户登录名查找用户

接口说明：根据用户登录名查找用户。

URL：http://localhost:8080/rrwb/crud/user/findUserByName。

HTTP 请求方式：POST/GET。

请求参数：如表 A-7 所示。

表 A-7　根据用户登录名查找用户的请求参数

参 数 名 称	类　　型	说　　明
login_name	String	登录名，必输

响应参数：如表 A-8 所示。

表 A-8　根据用户登录名查找用户的响应参数

参 数 名 称	类　　型	说　　明
code	Number	0：插入失败 1：插入成功

A.5　根据用户昵称查找用户

接口说明：根据用户昵称查找用户。

URL：http://localhost:8080/rrwb/crud/user/ findUserByNickName。

HTTP 请求方式：POST/GET。

请求参数：如表 A-9 所示。

表 A-9　根据用户昵称查找用户的请求参数

参 数 名 称	类　型	说　明
nick_name	String	昵称，必输

响应参数：如表 A-10 所示。

表 A-10　根据用户昵称查找用户的响应参数

参 数 名 称	类　型	说　明
code	Number	0：插入失败 1：插入成功

A.6　用户登录

接口说明：根据用户名和密码验证用户登录。

URL：http://localhost:8080/rrwb/crud/user/login。

HTTP 请求方式：POST/GET。

请求参数：如表 A-11 所示。

表 A-11　用户登录的请求参数

参 数 名 称	类　型	说　明
login_name	String	登录名，必输
pwd	String	密码，必输

响应参数：如表 A-12 所示。

表 A-12　用户登录的响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
data	String	查询到的数据的数组
rows	String	查询到的记录总数量

A.7　根据用户昵称模糊查找用户

接口说明：根据用户昵称模糊查找用户，并可以分页查询。

URL：http://localhost:8080/rrwb/crud/user/findUserLikeNickName。

HTTP 请求方式：POST/GET。

请求参数：如表 A-13 所示。

表 A-13　根据用户昵称模糊查找用户的请求参数

参 数 名 称	类　型	说　明
nick_name	String	昵称，必输
_id	Number	用户 ID，必输
page	String	当前页数，从 1 开始
size	String	每页显示的数量

响应参数：如表 A-14 所示。

表 A-14　根据用户昵称模糊查找用户的响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	String	查询到的数据的数组
rows	String	查询到的记录总数量

例如：

{"code":1,"pages":1,"data":[{"_id":103,"login_name":"1@1.com","gender":"female","nick_name":"hello","pwd":"111111","status":true,"timestamp":1427088263009},{"_id":92,"login_name":"wyy@163.com","gender":"female","nick_name":"wuyuan","pwd":"111111","status":true,"timestamp":1426058627885},{"_id":91,"login_name":"zp@163.com","gender":"female","nick_name":"zhangsan","pwd":"111111","status":true,"timestamp":1426058555856}],"rows":3}

A.8　上传头像缩略图

接口说明：上传头像缩略图。

URL：http://localhost:8080/rrwb/crud/uploadServlet。

HTTP 请求方式：POST/GET。

enctype：multipart/form-data。

请求参数：如表 A-15 所示。

表 A-15　上传头像缩略图的请求参数

参 数 名 称	类　型	说　明
x1 y1	Number	最初选择区域的左上角坐标
x2 y2	String	最初选择区域的右上角坐标

上传图片的 form 表单中，enctype 属性设置为"multipart/form-data"，以二进制格式上传图片。

响应参数：如表 A-16 所示。

表 A-16　上传头像缩略图的响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
url	Number	返回的头像缩略图的 url 地址

A.9　上传图片微博

接口说明：上传微博图片。

URL：http://localhost:8080/rrwb/crud/ uploadWeiboServlet。

HTTP 请求方式：POST/GET。

enctype：multipart/form-data。

请求参数：无。

响应参数：如表 A-17 所示。

表 A-17　上传图片微博的响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
url	Number	返回图片的 url 地址

A.10　发布微博

接口说明：发布微博。

URL：http://localhost:8080/rrwb/crud/weibo/add。

HTTP 请求方式：POST/GET。

请求参数：如表 A-18 所示。

表 A-18　发布微博的请求参数

参 数 名 称	类　　型	说　　明
user_id	Number	用户 ID，必输
user_nickName	String	
user_head_img	String	
img	String	
content	String	必输

响应参数：如表 A-19 所示。

表 A-19　发布微博的响应参数

参 数 名 称	类　　型	说　　明
code	Number	0：插入失败 1：插入成功

A.11　查询较早发布的微博

接口说明：查询较早发布的微博。

URL：http://localhost:8080/rrwb/crud/weibo/previous_weibos。

HTTP 请求方式：POST/GET。

请求参数：如表 A-20 所示。

表 A-20　查询较早发布的微博的请求参数

参 数 名 称	类　　型	说　　明
user_id	Number	用户 ID，必输
page	String	当前页数，从 1 开始
size	String	每页显示的数量
last_More_Time	Number	指定从哪个时间节点向后查询微博

响应参数：如表 A-21 所示。

表 A-21　查询较早发布的微博的响应参数

参 数 名 称	类　　型	说　　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	Array	查询到的数据的数组
Rows	String	查询到的记录总数量

A.12　查询尚未加载的最新发布的微博

接口说明：查询尚未加载的最新发布的微博。

URL：http://localhost:8080/rrwb/crud/weibo/latest_weibos。

HTTP 请求方式：POST/GET。

请求参数：如表 A-22 所示。

表 A-22　查询尚未加载的最新发布的微博的请求参数

参 数 名 称	类　型	说　明
user_id	Number	用户 ID，必输
page	String	当前页数，从 1 开始
size	String	每页显示的数量
currentTime	Number	上次加载最新微博的时间

响应参数：如表 A-23 所示。

表 A-23　查询尚未加载的最新发布的微博的响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	Array	查询到的数据的数组
rows	String	查询到的记录总数量

A.13　增加关注

接口说明：关注功能。

URL：http://localhost:8080/rrwb/crud/fans/add。

HTTP 请求方式：POST/GET。

请求参数：如表 A-24 所示。

表 A-24　增加关注的请求参数

参 数 名 称	类　型	说　明
fans_id	Number	当前用户 ID
followed_id	Number	被关注用户 ID

响应参数：如表 A-25 所示。

表 A-25　增加关注的响应参数

参 数 名 称	类　型	说　明
code	Number	0：插入失败 1：插入成功

A.14　查看当前用户是否已经关注某个用户

接口说明：查看当前用户是否已经关注某个用户，返回结果是一个对象，如果对象的 rows 不等于 0，则当前用户已经关注过某个用户。

URL：http://localhost:8080/rrwb/crud/fans/find。

HTTP 请求方式：POST/GET。

请求参数：如表 A-26 所示。

表 A-26　查看当前用户是否已经关注某个用户的请求参数

参 数 名 称	类　型	说　明
fans_id	Number	当前用户 ID
followed_id	Number	被关注用户 ID

响应参数：如表 A-27 所示。

表 A-27　查看当前用户是否已经关注某个用户的响应参数

参 数 名 称	类　型	说　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	Array	查询到的数据的数组
rows	String	查询到的记录总数量

A.15　分页查找当前用户的粉丝

接口说明：分页查找当前用户的粉丝。

URL：http://localhost:8080/rrwb/crud/fans/fanspage。

HTTP 请求方式：POST/GET。

请求参数：如表 A-28 所示。

表 A-28　分页查找当前用户的粉丝的请求参数

参 数 名 称	类　型	说　明
followed_id	Number	当前用户 ID
page	String	当前页数，从 1 开始
size	String	每页显示的数量

响应参数：如表 A-29 所示。

表 A-29　分页查找当前用户的粉丝的响应参数

参 数 名 称	类　　型	说　　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	Array	查询到的数据
rows	String	查询到的记录总数量

例如：

{"code":1,"pages":1,"data":[{"_id":16,"login_name":"1387654321","nick_name":"11","pwd":"111111","status":true,"timestamp":1419917453214,"birthday":"2014-12-19","qq":"","provinceValue":"7","cityName":"","msn":"msss","real_name":"aa","provinceName":"吉林省","blog":"c","cityValue":"","img":"3.jpg","other_email":"","age":50}],"rows":1}

A.16　分页查找当前用户关注的用户

接口说明：分页查找当前用户关注的用户。

URL：http://localhost:8080/rrwb/crud/fans/followspage。

HTTP 请求方式：POST/GET。

请求参数：如表 A-30 所示。

表 A-30　分页查找当前用户关注的用户的请求参数

参 数 名 称	类　　型	说　　明
fans_id	Number	当前用户 ID
page	String	当前页数，从 1 开始
size	String	每页显示的数量

响应参数：如表 A-31 所示。

表 A-31　分页查找当前用户关注的用户的响应参数

参 数 名 称	类　　型	说　　明
code	String	0：插入失败 1：插入成功
pages	Number	查询到的总页数
data	String	查询到的数据的数组
rows	String	查询到的记录总数量